conference

proceedings

Special Workshop on Intelligence at the Network Edge

San Francisco, CA, USA
March 20, 2000

Sponsored by
USENIX
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Contents

Special Workshop on Intelligence at the Network Edge

March 20, 2000
San Francisco, California, USA

Session 1 — QoS at the Edge

Smartbox Architecture 1
Bülent Yener, Bell Laboratories

Integrating Active Networking and Commercial-Grade Routing Platforms 11
R. Jaeger and S. Bhattacharjee, University of Maryland; J. K. Hollingsworth, R. Duncan, T. Lavian, and F. Travostino, Nortel Networks

Session 2 — Addressing at the Edge

RSIP: Address Sharing with End-to-End Security 20
Michael S. Borella, 3Com Corp.; Gabriel E. Montenegro, Sun Microsystems Laboratories

Session 3 — Content Caching at the Edge

Towards a Platform for Intelligent Activity at the Edge 29
Hilarie Orman, Novell, Inc.

Smart Box Architecture

Bülent Yener *

Abstract

Fundamentally the IP-based networking is designed for delivering data traffic with best-effort service, thus it is not capable of providing end-to-end QoS. Several architectures have been proposed for providing QoS in the Internet: The *integrated services* (Intserv) model is based on reservations and can provide QoS, however; it is not scalable. The *differentiated services* (Diffserv) approach is scalable but falls short of ensuring deterministic guarantees —in particular for the services that belong to the same class. Finally, the *multi protocol label switching* (MPLS) architecture provides mechanisms for QoS-based routing but does not have the necessary resource management and scheduling support to ensure it.

This work proposes a hybrid solution which combines the best of these technologies. First, at the network boundary Diffserv like Service Level Agreements (SLA) are provided to users by intelligent edge routers called the **SBoX servers**. An SBoX server uses Class Based Queuing (CBQ) with a hierarchy of flow aggregation. At the top a *commodity-flow* is defined for the aggregate flow between a pair of egress points. The packets of the same commodity-flow are marked by an MPLS label, which is globally unique within an Autonomous System (AS). Each commodity flow is partitioned to a set of *macro-flows* which are offered to users as SLAs. An SBoX server manages macro-flows and commodity flows only, and leaves the management of each macro-flow (at the micro-flow level based on some policies) to the enterprise/users which signed the SLA. Second, the commodity-flows are managed and supported inside the network by an add-on Label Switching Router (LSR) called the **SBoX router** which performs MPLS of commodity-flows with CBQ. The main reason for an add on solution is the lack of end-to-end deployment of LSRs, and the vertically integrated architecture of the legacy routers. This paper explains the SBoX architecture and reports experimental results obtained on a prototype network.

*Bell Laboratories, Lucent Technologies, 700 Mountain Ave., Murray Hill, NJ 07974 E-mail: yener@research.bell-labs.com, Tel: (908) 582 7087

1 Introduction and Motivations

As the Internet gets commercialized, the need for providing QoS becomes imminent. Current infrastructure of the Internet cannot support QoS since it is designed for best-effort service model. Its routers operate with FIFO scheduling without any guarantees. Increasing the network capacity by adding more bandwidth and routers is not always feasible or efficient. The network must have mechanisms to distinguish QoS requirements of different applications that share the same infrastructure and process them accordingly.

There are two fundamentally different perspectives to the QoS problem in the Internet: (i) *Integrated Services* (Intserv), and (ii) *differential services* (Diffserv). The Intserv approach [SPG97, Wro97, SW97] requires high-end routers to maintain per-micro-flow [1] state information and to perform complex link scheduling algorithms. As the number of flows increase and/or change frequently, the overhead of this approach does not scale.

The Diffserv approach aims to [Cla97, CW97, NJZ97, SZ98, ea98] to reduce the per-flow complexity by providing an aggregated treatment of user traffic that belongs to the same service class. Packets in the Diffserv model are marked, at the network entry points, to indicate whether or not the source follows its SLA. The packets that violate their SLA (i.e., OUT packets) are dropped with a higher probability than the packets obey their SLA (i.e., IN packets). Several performance studies [BW99, IN98] of Diffserv approach show that (1) it cannot offer a quantifiable service to TCP traffic, (2) there is strong dependency between IN and OUT packets (due to shared queue) and consequently IN packets may be dropped, and (3) mixing IN and OUT packets in a single TCP connection reduces the connection's performance. Non-deterministic and non-quantifiable QoS cannot be acceptable for certain applications, such as IP telephony, real-time interactive transactions, IP high definition video (e.g., HDTV), and Virtual Private Networking (VPN) for which pricing and QoS are strongly coupled.

Other approaches such as Core-Stateless Fair Queu-

[1] A micro-flow is a application level flow defined by its source, destination address, port numbers and protocol identifier.

ing (CSFQ) [SSZ98] attempt to compromise by replacing the per flow state overhead at the routers by Dynamic Packet States (DPS) [ea99b]. The DPS approach requires checking and updating a state information associated with each packet. Thus it introduces extra processing complexity at each node. Furthermore it requires modification of existing routers.

Recently, Multi Protocol Label Switching (MPLS) [RVC98] has become an attractive technology. The main contributions of MPLS are twofold. First, it decouples routing from forwarding. Thus, *explicit routing*, based on metrics different from the ones used by the traditional routing algorithms can be deployed. Second, it aggregated the flows to Forwarding Equivalent Classes using virtual circuit switching. The former enables traffic engineering and QoS based constrained routing while the latter provides a scalable solution. Although it contributes significantly, the MPLS is not enough for providing QoS since it does not address resource allocation, scheduling and admission control problems.

1.1 Principles of the Solution

Proposals to the QoS problem in the IP networks must address the fundamental trade off between the scalability and the QoS guarantees. In order to provide end-to-end QoS guarantees, some notion of resource reservation is necessary. However, such reservation must be for aggregated flows to minimize per-flow state information at the routers for a scalable solution. The granularity of aggregation (i.e., the set of the flows that are treated by the network uniformly) impacts on the level of service each micro-flow receives and may create fairness problems. Furthermore, real-time, adaptive resource management and admission decisions increase the complexity of the interior nodes thus cause performance bottlenecks.

From these observations a separation of the QoS functions at the edge and at the core nodes can be identified as follows. Processing and state information overhead that cause scalability problem at interior nodes should be pushed to the egress points. Thus, edge nodes must be in charge of (i) admission control, (ii) QoS based network access which includes micro-flow management, traffic engineering, and generating aggregate flows, and (iii) service based billing and charging.

In contrast, the responsibility of interior nodes must be limited to a minimum set of operations sufficient to support the egress QoS mechanisms inside the network. Their complexity can be reduced by adapting the following principles: (i) separating routing from forwarding, (ii) operating on a pseudo circuit switching (e.g., MPLS, virtual circuit switching) mode to eliminate per-packet IP lookup and filtering overhead (iii) using aggregated flow management with CBQ or WFQ scheduling, and (iv) using light-weight signaling protocols for aggregate flow reservations.

These goals may be achieved by a hybrid architecture that combines the best of Intserv, Diffserv, and MPLS proposals with QoS provisioning and admission control.

1.2 Overview of the Proposed Solution

This work proposes such a hybrid solution. First, at the network boundary intelligent edge routers called the **SBoX servers** are deployed. An SBoX server combines the Diffserv and MPLS architectures at a network access point. SBoX servers provide Diffserv like SLA. Each agreement is a macro-flow with a specific bandwidth, and established between two egress points. A macro-flow is shared by all the application-level micro-flows between these end points. All the macro-flows in the SLA are aggregated to a *commodity-flow*. The packets of the same commodity-flow are marked by an MPLS label which is globally unique within an AS. The SLA's are enforced and packed into a commodity-flow by CBQ scheduling.

Second, inside the network, a subset of the best-effort IP routers are enhanced with a programmable, add-on Label Switching Router (LSR) called the **SBoX router** (SBoX). An SBoX router performs MPLS with CBQ over IP, in a totally transparent way to current IP-based network infrastructure and protocols. It controls some of the links of such IP nodes, so that it switches labeled traffic (commodity-flows) directly, and passes other traffic (best effort) to the legacy router. The label switching is done by using the global labels and commodity-flows are protected from each other and from best-effort traffic using CBQ. The main motivation behind add-on SBoX routers is to address the following problems: (1) the lack of end-to-end deployment of LSRs (i.e., many routers are not MPLS capable), and (2) the lack of programmability of the legacy routers. An add-on approach provides service guarantees in a network that contains legacy routers without replacing or changing them.

Third, new algorithms are proposed for provisioning, resource allocation, and admission control. The core of provisioning and resource allocation problem is the placement of SBoX routers and the selection of the links that enforce QoS guarantees. The SBoX routers and the links that they manage induce an virtual premium network which is used for label switching of the QoS traffic.

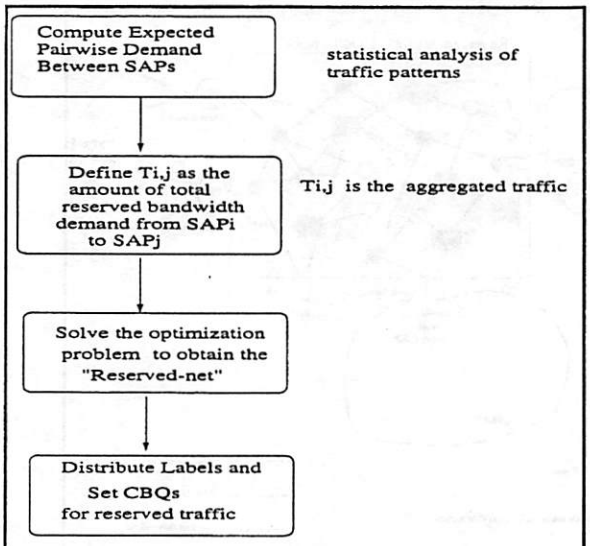

Figure 1: Steps for Provisioning and Configuration of QoS Traffic

Given the pairwise statistical demands for QoS traffic over a fixed interval, we formulate the problem designing an optimal virtual network as a multi-commodity problem with integer variables. The objective function is to *pack* the commodity-flows into a minimal feasible subgraph while avoiding bandwidth fragmentation on the links. An SBoX router is associated (activated) with each node that appears in the solution of this optimization problem. The solution to the optimization problem specifies the amount of bandwidth allocation necessary for each commodity. Bandwidth allocation and protection is done by setting CBQ mechanisms in the involved SBoX routers. The parameters of the CBQ scheduling and the labels of macro-flows are signaled with a light weight protocol which aims to minimize the communication overhead.

This paper is organized as follows. In Section 2 we address provisioning and configuration issues. Section 3 presents the components of the SBoX architecture. In Section 4 we explain the prototype implementation and experimental results. Finally we conclude in Section 5.

2 Network Provisioning

We consider the IP-based Internet as an interconnection of autonomous systems in which a provider has control of all the resources (i.e., it is a single administrative domain). An AS can be accessed through some specific entry points called *Service Access Points* (SAPs). A provider aggregates the user traffic at the SAPs. The network can be connected to the other networks via border routers (BRs) that are treated as SAPs in our model. SAPs are involved with admission control and perform policy based bandwidth management at network boundary. The discussion in this work is focused on a single AS.

We distinguish between two types of traffic carried by the network: (1) reserved or committed QoS traffic, and (2) non-reserved or best-effort traffic. Network resources are provisioned and configured to accommodate to the QoS traffic. Thus, provisioning and configuration require estimating pairwise bandwidth demands for QoS traffic between all SAPs and BRs in an AS. There are several techniques for source modeling that can be adapted for this purpose [ML97]. Provisioning and configuration of QoS traffic in the SBoX architecture are achieved by creating a virtual premium network called the **reserved-net**. The reserved-net is a virtual dedicated network that connects all the SAPs and BRs. It is used to (i) protect the reserved traffic from the best effort traffic, and (ii) forward QoS traffic using MPLS. The best-effort traffic may be permitted to use the reserved-net in addition to the second subnetwork; however, the reserved traffic has preemptive priority. Figure 1 depicts the main steps for provisioning and configuration of QoS traffic.

We note that the problem of provisioning a reserved-net is a version of network optimization problem for which a rich literature exists (e.g., [Dov91a, Dov91b, CFZ94, ACL94, Ash95, For96, Lee95, MMR96]). However, the objective function of our problem is different as we explain in the next section.

2.1 Optimum Virtual Network

Network is represented by a graph $G = (V, E)$ where node set V is partitioned into two subsets. Set R contains the routers, set S contains the SAPs. End-points of a commodity-flow belong to the set S. The set E contains the network links, each of which has capacity (bandwidth) $c_{(i,j)}$ (bits/sec) $\forall (i,j) \in E$. For simplicity we assume a symmetric model so that each direction of a link has the same capacity. The reserved-net is a subgraph $G_r \subset G$, represented as $G_r = (R', S, E')$. Let T be a traffic matrix in which an entry $t_{i,j}$ indicates the aggregated bandwidth request for QoS traffic from i to j $\forall i, j \in S$. In other words $t_{i,j}$ is the sum of the bandwidth of macro-flows from SAP i to SAP j. We assume that T is based on the statistical information capturing a correlated source behavior (e.g., the traffic volume between 8-11 AM). The optimization problem can be formulated as an instance of mixed-integer

multi-commodity flow problem, where a commodity k is defined for a pair of nodes $k, l \in S$ such that $t_{k,l} > 0$ and $k \neq l$. In other words, a commodity k is provided from each SAP k to another SAP l, so that k has non-zero bandwidth request to l.

Our cost measure is to *pack* the *routes* of reserved traffic together so that (i) number of links used by the solution is minimum, and (ii) **bandwidth fragmentation** is minimized for each link included into the reserved-net. There are two motivations behind using bandwidth fragmentation as a measure: (1) support policies for sharing of link bandwidth between best-effort and QoS traffic (i.e., as a part of QoS provisioning), and (2) minimize the amount of "unused" or "left-over" bandwidth of a reserved link which is smaller than any expected commodity-flow bandwidth request, if no best-effort traffic is allowed on the reserved-net.

In the formulation, we have a threshold value $\delta_{i,j}$ for each link. If unused or left-over bandwidth is less than $\delta_{i,j}$ then the variable $y_{i,j}$, which is 1 only if link (i,j) is used, contributes to the penalty term in the objective function with a coefficient α as shown in Figure 3. Note that α can be chosen to control the impact of bandwidth fragmentation on the cost function.

The variables and parameters of the optimization problem are as follows:

- Flow variable $f^k_{i,j}$ which takes a real value and indicates the amount of commodity k over the link (i,j).
- A 0-1 integer variable $x_{i,j}$ which indicates the link (i,j) is used by any commodity. It is 1 if used, 0 otherwise. In other words, $x_{i,j}$ denotes whether link (i,j) is included in the reserved-net.
- A 0-1 integer variable $y_{i,j}$ which indicates whether or not the *unused* bandwidth of the link (i,j) is below a given *fragmentation threshold* $\delta_{i,j}$.

Constraint (1) ensures the flow balance for each commodity. If the node is an intermediate node (i.e., $i \in R$ is a router), the in-flow should be equal to the out-flow. Otherwise, the node i is a macro-flow end-node and the flow into this node from commodity k is $t_{k,i}$ for each commodity k. The inequalities (2) and (3) ensure that total load on a link (i,j) does not exceed its capacity. Furthermore, the load is assigned such that the bandwidth fragmentation is minimized by the penalty term $y_{i,j}$. Inequality (3) ensures that the penalty term exists only for the links in the reserved-net. Inequalities (4) and (5) are self explanatory.

The solution to this optimization problem is a set of $x_{i,j}$ and $y_{i,j}$ values, that minimize the cost function while connecting each pair of egress points. The links with $x_{i,j} = 1$ are called **reserved-links** and the nodes incident to reserved-links are called **reserved-nodes**.

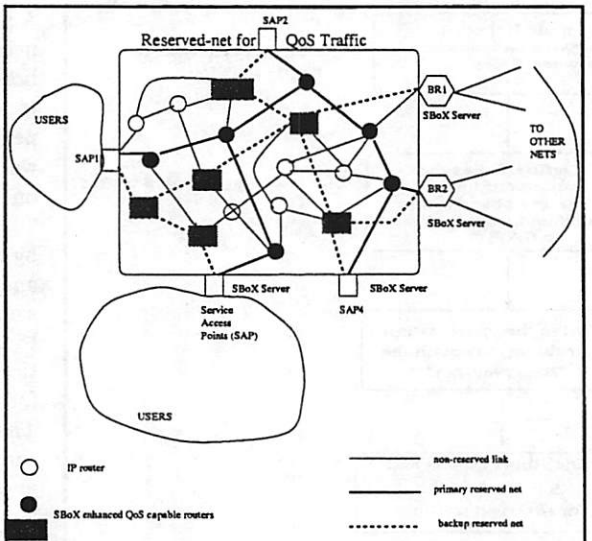

Figure 2: Reserved-Net: Virtual network for MPLS traffic.

Minimize: $C = \sum_{i,j} x_{i,j} + \alpha(y_{i,j}) \quad \forall (i,j) \in E$ and $\forall k$

Constraints:
s.t.

(1) $\sum_{j \neq i} f^k_{(j,i)} - \sum_{j \neq i} f^k_{(i,j)} = \begin{cases} t_{k,i} & \forall k, i \ s.t. \ k \neq i \\ 0 & \forall i \in R \end{cases}$

(2) $\sum_k f^k_{i,j} \leq (c_{i,j} - \delta_{i,j}) x_{i,j} + \delta_{i,j} y_{i,j} \quad \forall (i,j) \in E$

(3) $y_{i,j} \leq x_{i,j}$

(4) $x_{i,j} \in \{0,1\} \quad \forall (i,j) \in E$

(5) $f^k_{i,j} \geq 0$

Figure 3: Mixed-integer formulation of the problem

The union of the reserved-links induces a connected graph G_r which is our reserved-net as shown in Figure 2.

A path from SAP_k to SAP_l on the reserved-net is called **reserved-path**. It is the label switched path (LSP) that carries the commodity-flow for commodity k. It may be desirable to over allocate the capacity for providing fault-tolerance and dynamic admission control. A simple way to achieve this is to add slack variables to the $t_{i,j}$ and solve the above optimization problem. Note that the exact solution of the above optimization problem is computationally expensive; thus we may either use heuristics to obtain suboptimal solutions, or solve the optimization problem off-line and store it for different traffic matrices T in advance. Since our focus is on the QoS aspects of the problem, we will

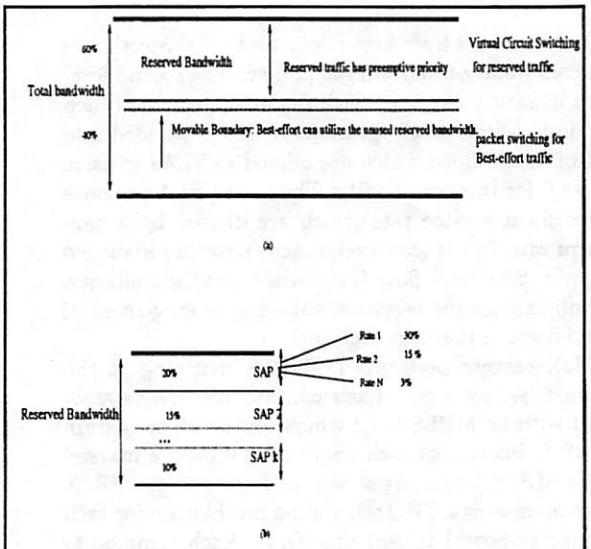

Figure 4: Bandwidth Sharing

not further elaborate on the optimization issues.

2.2 Resource Allocation

The solution to the optimization problem determines reserved-links and the amount of flow that can be assigned to these links. In particular the $f_{i,j}^k$ value indicates the bandwidth request of commodity-flow on link (i,j) that resides in the reserved-path from SAP_k to SAP_l. In order to enforce service guarantees, $f_{i,j}^k$ must be reserved and protected on link (i,j). Resource allocation in the reserved-net is done at the commodity-flow level and enforced by CBQ.

Different policies may be adapted to adjust the interaction between the reserved and the best-effort traffic. We propose a **movable boundary** scheme, in which the best-effort can use the available reserved bandwidth, where reserved traffic has preemptive priority as shown in Figure 4. The position of the boundary can be dictated by the network economics and utilization. For example, an ISP may decide to reserve only 60% of link capacity to the aggregated reserved-traffic.

The policy for setting such a boundary can be enforced in the optimization problem by the threshold variable $\delta_{i,j}$ and the coefficient α.

Distribution of MPLS labels and setting the CBQs can be done using a simple signaling protocol as presented in the next section.

2.3 Light Weight Signaling Protocol

Currently two protocols are being discussed to set a LSP (i.e., to distribute the labels and make the reservations at LSR): RSVP extension, and CR-LDP [ea99a]. The RSVP has the softstate overhead and lacks a reliable protocol (it is based on IP) for distribution. Thus, its response delay is higher. In contrast, the CR-LDP uses TCP for reliable delivery but is subject to all the problems associated with the performance of TCP. This section presents a generic signaling protocol, based on 3-way handshake, with the following steps:

1. SAP_x makes a request of R bps to a SAP_y for the corresponding commodity-flow by
   1.1 sending $REQ(Label, R, seq\#)$ message.
   1.2 starts a timer for reply.
2. each SBoX router on the reserved-path
   2.1 marks the forwarding table for rate R and $Label$ and
   2.2 appends its ID to the REQ message.
   2.3 marking is not a commitment but a tentative state. It is associated with a timer and if no reply (REP) message is received, then there will be timeout and the tentative commitment will be reset.
3. upon receiving the REQ message, the SBoX server at SAP_y
   3.1 sends $REP(Label, R, seq\#)$ message back by reversing the reserved-path recorded into the REQ message.
   3.2 sets a timer for confirmation (CONF) message
4. each SBoX router on the path
   4.1 identifies the REQ/REP using the Label,
   4.2 *commits* for the rate R that REP message carries,
   4.3 unmarks the REQested rate
   4.4 starts a timer for confirmation
5. upon receiving the REP message, the SAP_x confirms the reserved-path by
   5.1 sending a $CONF(Label, R)$ message to SAP_y or
   5.2 starts transmitting data.
6. if the timer in SAP_x times out for the REQ message then it resends a REQ message with an incremented seq #.
7. if an SBoX router receives a redundant REQ message and
   7.1 it has marked the forwarding table then assumes that previous REQ got loss at the segment between itself and SAP_y
   7.2 it has also committed then assumes that REP got loss at the segment between itself and SAP_x
   7.3 takes no action and waits the edge routers to response.
8. upon committing to a REQ message if an SBoX router receives no CONF message or data then it time outs and releases the commitment.
9. each router maintains the commodity-flow until a TEARDOWN message is received.

Notice that QoS requirement is expressed simply as the amount of the bandwidth. The traffic specifications such as burst size burst length are omitted since they are controlled at the egress points by SBoX servers. Furthermore, constructing a virtual premium network (i.e., the reserved-net) ensures that for any reserved-link (i,j), $\sum_k f_{i,j}^k \leq c_{i,j}$.

3 SBoX Architecture

The SBoX architecture comprises two components: (1) SBoX servers, and (2) SBoX routers. The SBoX servers involve with admission control and management at the network edge. SBoX routers are the interior network nodes that perform only label switching of macro

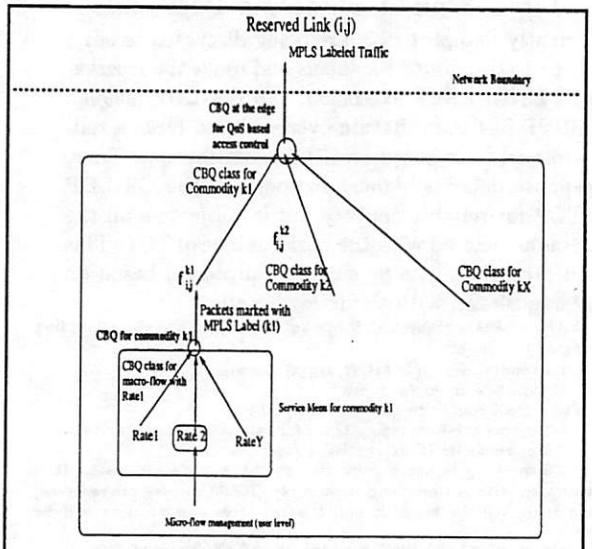

Figure 5: CBQ hierarchy and labeling maintained by an SBoX Server

flows for QoS based routing. We will describe the SBoX servers and routers in detail in this section.

3.1 SBoX Servers

The SBoX architecture deals with the complexity of QoS support by delegating it to the SBoX servers. An SBoX server is an **intelligent edge-router** that resides at network egress point (i.e., SAP) and provides QoS based network access. A SBoX server is a programmable device with an open architecture, and it is in charge of managing the resources and requests from SAPs. It separates QoS traffic from best-effort and labels the QoS traffic for MPLS routing inside the network. An SBoX server uses hierarchical traffic management mechanisms which can be configured dynamically to adapt to the changing policies of an ISP. We note that since the adjacent AS are treated as SAPs, their resource management is also handled by SBoX servers.

3.1.1 Flow Management and Labeling

SBoX server is a configurable device which has hierarchical traffic management capability. Figure 5 depicts the hierarchical management of a reserved-link (i, j) by an SBoX server. At the top of the hierarchy, the bandwidth of (i, j) is shared by multiple commodity-flows whose bandwidth allocation is determined by the optimization problem presented in section 2.1. Link (i, j) needs to be managed to ensure: (1) protection of commodity-flows from best effort, and (2) protection of each commodity-flow from each other. The second level of the hierarchy concerns with the management of each commodity-flow. Each commodity-flow is divided into a set of macro-flows which are offered as SLAs to users of a SAP for this commodity. Thus, each SLA provides a predefined service rate which are offered by a **service menu**. In the last level, macro-flows are managed in application level flow (i.e., micro-flow) granularity. We will explain the service menus and management of macro-flows in the next section.

SBoX servers performs IP-MPLS mapping at the commodity-flow level. Each commodity flow is associated with an MPLS label which can be global within the AS [2]. Packets of each commodity-flow are marked by the MPLS label. As shown in Figure 5, the SBoX server maintains a CBQ scheduling mechanism for each commodity-flow that uses edge (i, j). Each commodity is considered as a class and classes are strictly isolated from each other (i.e., fire-walled). The resource sharing between best-effort and QoS traffic is based on the movable boundary scheme as explained above. Packets of each commodity are filtered and scheduled based on their MPLS labels.

An SBoX server also maintains billing and accounting information at different levels of granularity. It can provide call description record (CDR) type of information at the commodity-flow or at the macro-flow level. We will explain billing and charging issues at the next section further.

3.1.2 Admission Control with SLAs

The committed traffic is accepted to the network through admission control, which ensures that bandwidth allocation satisfies the service descriptions of user requests. An important part of admission control is monitoring and managing the *available bandwidth* for each commodity.

The SBoX architecture provides various data rates called **service rates**, that are a multiple of a base rate (e.g., 1Mbps, 5Mbps, 10Mpbs, 20Mbps) in a service menu. Each data rate represents a different SLA. Users sign a contract with their provider by picking an SLA from a service menu.

As shown in Figure 5, each SLA corresponds to a macro-flow. It is left to the users/enterprises to manage the access to the pipe. There are several commercial products available to differentiate business critical

[2] Note that 20 bits from the 4-byte MPLS label can support 2^{20} commodities which corresponds to approximately 10^3 SAPs within an ASP. If the number of SAPs exceeds such number then labels must be swapped and cannot be global.

traffic based on enterprise policies (e.g., Xedia's Access-point, Packeteer's Packetshaper, Checkpoint's Flood Gate, Allot's AC 200-330).

The challenge is how to decide which service rates to offer and how many requests to grant for each (macro-flow) rate. We believe that the answer should have low overhead and also address the network economics. For example, each service rate in the service menu can be assigned a number of **tokens** (permits) during provisioning. The weighted sum of the tokens is equal to the total amount of reserved traffic that the provider is willing to accept (where the weights represent the data rates). Each SAP and BRs are given a set of tokens based on (i) the expected number of connections, and (ii) the bandwidth demands between them. Thus, each SAP/BR can make a local decision how much traffic to accept to the reserved-net using the token/permit pool it has. For example, lets refer to Figure 5. Suppose that the SAP is given x tokens of 512Kbps and y tokens of 1Mbps for commodity k_1. Then this SAP can accept at most $x + y$ requests for this commodity such that $x512 + y1000 \leq f_{i,j}^{k_1}$ kbps.

There are two advantages of providing SLAs with token based admission control. First, it reduces the traffic management overhead at the commodity-flow level. Since users must pick an SLA from a service menu in advance, the policing, shaping and scheduling in the network can be set ahead of time, even before the SLAs are offered. In contrast, if we replace the predetermined rates with a dynamic management scheme then requests for macro-flows must be accommodated on demand basis. This requires setting the traffic filters and shapers on demand as well. Thus, it may increase the processing time for requests and response delay. Second, it provides efficient billing and accounting mechanism to an ISP. For example, each token can have a price based on the rate that it is associated with. Thus charging can be reduced to the token holding time multiple with the price of this token. The price of a token can be changed based on the demand for bandwidth. For example a discount price can be offered during off-peak hours while price can be increased during peak hours.

3.2 SBoX Routers

An SBoX router is a programmable, stand-alone, label switching router with a limited number of network interfaces. An SBoX router cannot perform IP-based routing and its routing capability is limited only to label switching with CBQ scheduling.

Since the legacy routers are vertically integrated closed boxes, an SBoX router may be deployed as an add-on device [3]. An SBoX router controls the reserved links, identified by the reserved-net, incident to a legacy router, as shown in Figure 6.

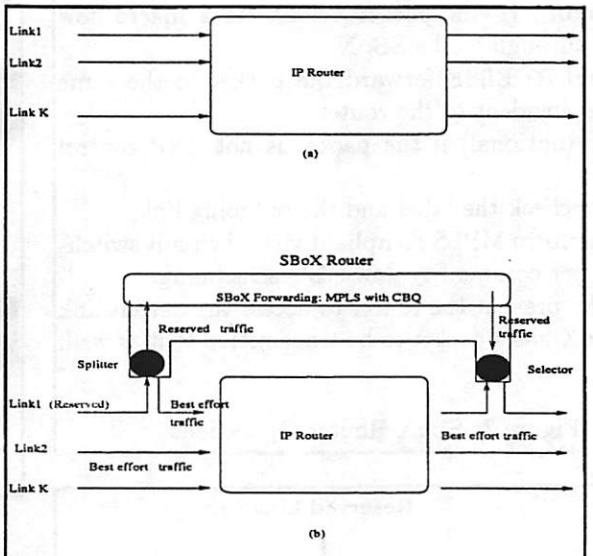

Figure 6: SBoX Operation: QoS traffic has precedence over best-effort traffic. Best-effort traffic uses either links that are not a part of the reserved-net or the idle portion of reserved link. Combination of SBoX and best-effort router provides functionality found in a *dual-switch* [Kum74, Zaf74, OY94, NBS95].

The number of network interfaces and the line speeds that an SBoX supports are dictated by the network economics and considered to be outside the scope of this paper. However such constraints can be accommodated into the formulation of the optimization problem to limit the number of reserved links incident to a legacy router that can be in the solution. The main functions of the SBoX are summarized in Figure 7.

Some of these functions are performed by two network interface modules called the **splitter**, and the **selector**. The splitter intercepts all incoming packets, examines their header information, and decides where to forward the packet (outgoing link for MPLS cut-through, or an input link of the best-effort router. The splitter performs the above **Fget**, **Fchk**, **FcuthruS**, **FcuthruR**, and **Fdrop** operations. The splitter uses a buffer to store the initial portion of incoming packets until the label is processed. If the packet does not carry

[3] Certainly, it would be the best if an SBoX can be transplanted into a legacy router however current legacy routers do not provide such modular and open architectures.

Fget: intercept the traffic into the *reserved node*;
- **Fchk:** check the header to determine its type;
- **FcuthruS:** IF the packet belongs to a macro flow then cut through to the SBoX;
- **FcuthruR:** ELSE forward the packet to the same input link incident to the router.
- **Fdrop:**(optional) if the packet is not a IP control packet.
- **Fnext:** check the label and the outgoing link;
- **Fvc:** perform MPLS compliant virtual circuit switching with per commodity-flow CBQ scheduling.
- **Fblock:** prevent the router to access the output link if the SBoX has a packet to be transmitted to it as well.

Figure 7: SBoX Router Operations

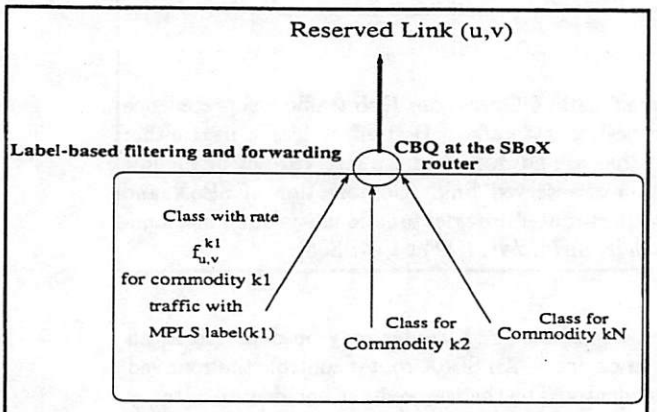

Figure 8: CBQ Class hierarchy in an SBoX router

a label, then it is considered as a best-effort, and it is forwarded immediately to the best-effort router without waiting and storing the rest of the packet. The size of the buffer is a function of the link rate and the splitter processing speed.

Contention occurs if best-effort router is allowed to forward packets to a reserved link. Thus we need to control the access to the output link. The selector is an arbitrator circuit, which ensures that as long as there is data in the SBoX's output buffer, the reserved link cannot be used by the legacy router. The splitter is functionally a a 2x1 multiplexer. The inputs of the multiplexer are the SBoX and the best-effort router. The multiplexer selects the SBoX as long as it has data. The splitter and selector reside inside the SBoX and their functions can be combined in an interface card with two ports and a limited processing capability.

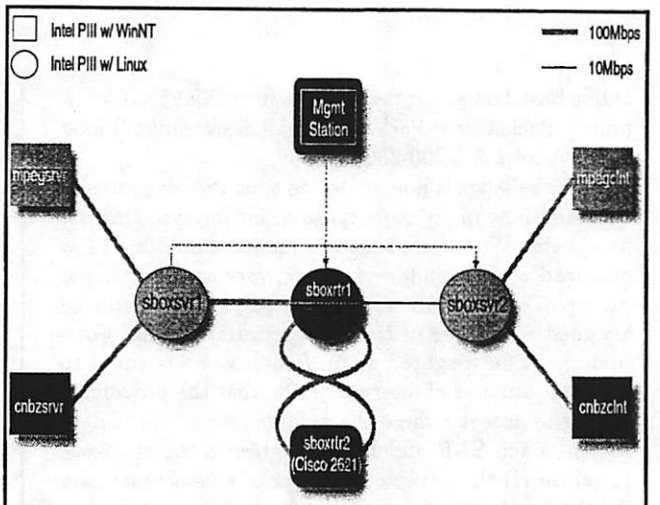

Figure 9: SBoX Prototype Testbed.

3.2.1 Flow Management and Forwarding

While the best-effort router performs IP-based routing, packet forwarding in an SBoX is based on label switching using MPLS. Upon entering the network, QoS traffic is handled exclusively by the SBoX routers. Thus it bypasses all best effort traffic. Each SBoX router has a **forwarding table**, associated with each link. An entry in the forwarding table indicates the outgoing link and the reserved bandwidth for a commodity-flow identified by its MPLS label.

The protection and enforcement of service guarantees is done with CBQ as shown in Figure 8.

4 SBoX Testbed Prototype

The testbed for SBoX project consists of seven Intel Pentium IIIs and a Cisco 2621 router connected with the topology depicted in Figure 9. Four of the seven PIIIs are running Windows NT, a pair of which for the Lucent MPEG video streaming (*mpegsrvr* and *mpegclnt*) and the other pair of which for the Lucent Cineblitz video streaming (*cnbzsrvr* and *cnbzclnt*). Three of the remaining PIIIs are running Linux with MPLS and CBQ support compiled into the kernel, two of which are the SBox servers (*sboxsvr1* and *sboxsvr2*) and one of which is an SBox router (sboxrtr1) managing commodity-flows inside the network. The Cisco 2621 router (sboxrtr2) plays the role of a conventional IP router for best-effort traffic.

One MPEG video stream of approximately 4Mbps flows from *mpegsrvr* to *mpegclnt*. Multiple cineblitz video streams, each about 1.5Mbps, flow from *cnbzsrvr* to *cbnzclnt*. These video streams are the premium

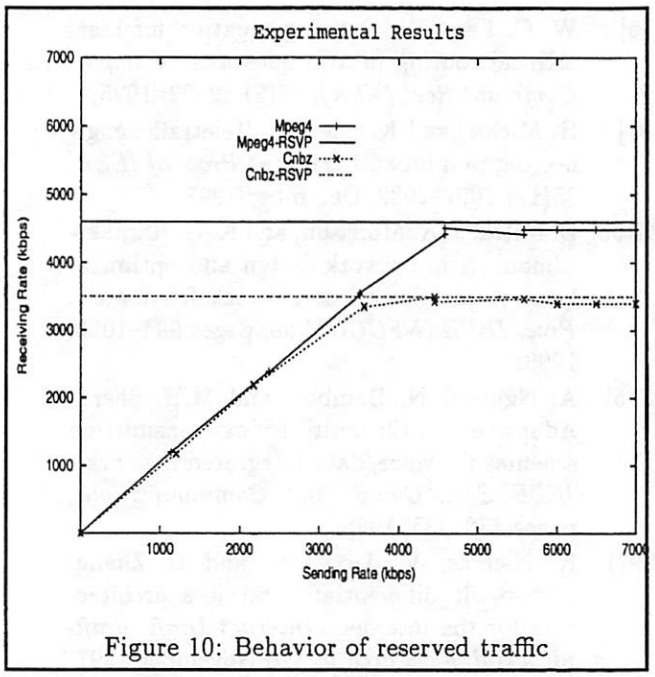

Figure 10: Behavior of reserved traffic

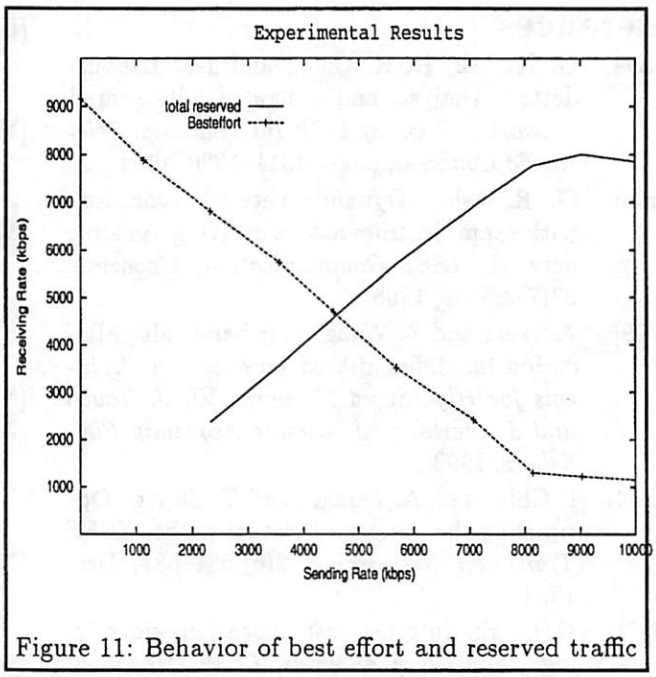

Figure 11: Behavior of best effort and reserved traffic

QoS traffic and can be selectively protected with guaranteed bandwidth. Premium traffic are label-switched across the network between *sboxsvr1* and *sboxsvr2*. A best-effort UDP traffic, with adjustable bandwidth consumption, flows from *sboxsvr1* to *sboxsvr2* as the background noise to create congestion. The noise traffic is not label-switched; it is routed through regular IP routing.

The noise traffic entering *sboxrtr1* is diverted to the Cisco router and then looped back to *sboxrtr1* before it is routed to its final destination. This is to simulate (1) the Split and Select operation, and (2) the ability of SBoX to inter-operate with existing IP routers without affecting the existing IP traffic. Note that all traffic goes through the bottleneck link, the 10Mbps link between *sboxrtr1* and *sboxsvr2*.

4.1 Experimental Results

Using the testbed implementation we conducted experiments to examine the performance of the SBoX architecture. The SBoX router *sboxrtr1* manages two commodity-flows and it is connected to the *sboxsvr2* with a full-duplex bottleneck link with capacity 10 Mbps as mentioned above.

We observe in Figure 10 that MPLS compliant QoS traffic for both commodities are protected from each other. Furthermore, their total sending rate is bounded by 8Mbps which is the total reserved bandwidth on this bottleneck link. Figure 11 shows that the best-effort traffic takes all the unused reserved bandwidth thus maintaining a high link utilization. As the total reserved traffic amount increases, the best-effort rate reduces to the unreserved portion of the link bandwidth. This behavior verifies that the *movable boundary* scheme works efficiently for providing high utilization. Furthermore it also shows that QoS traffic is protected from the best-effort.

5 Discussion

In this work we presented a hybrid architecture to provide deterministic QoS on the IP-based networks. The architecture combines the best features of Diffserv, Intserv, and MPLS proposals. At the core of the proposed architecture, precise provisioning and admission control algorithms are proposed. These algorithms determine a virtual premium network that is used to handle QoS traffic. The complexity of micro-flow management is done at the network boundary by intelligent edge routers called the SBoX servers. The service level agreements offered at the network boundary are supported by add-on label switching routers called the SBoX routers inside the network. A prototype testbed is implemented and experiments indicate that the proposed architecture delivers its features.

Acknowledgments

The author wishes to thank Eran Gabber for valuable discussions and to Gong Su for the implementation of the test bed.

References

[ACL94] G. R. Ash, K. K. Chen, and J-F. Labourdette. Analysis and design of fully shared networks. *Proc. of 14'th International Teletraffic Congress*, pages 1311–1320, 1994.

[Ash95] G. R. Ash. Dynamic network evolution, with examples from at&t's evolving dynamic network. *IEEE Communications Magazine*, 33(7):26–39, 1995.

[BW99] A. Basu and Z. Wang. Fair bandwidth allocation for differentiated services. *in Protocols for High Speed Networks VI, J. Touch and J. Sterbenz ed. Kluwer Academic Publishers*, 1999.

[CFZ94] I. Chlamtac, A. Farago, and T. Zhang. Optimizing the system of virtual paths. *IEEE Trans. on Networking*, 2(6):581–587, Dec. 1994.

[Cla97] D. Clark. Internet cost allocation and pricing. *Internet Economics, L. W. McKnight and J. P. Bailey (eds)*, pages 215–252, 1997.

[CW97] D. Clark and J. Wroclawski. An approach to service allocation in the internet. *Internet Draft, draft-clark-diff-svc-alloc-00.txt*, July 1997.

[Dov91a] R. D. Doverspike. Algorithms for multiplex bundling in a telecommunications network. *Operations Research*, 39:925–944, 1991.

[Dov91b] R. D. Doverspike. A multi-layered model for survivavility in intra-lata transport networks. *Proc. IEEE GLOBCOM'91*, pages 2025–2031, 1991.

[ea98] S. Blake et al. A framework for differentiated services. *draft-ietf-diffserv-framework-0.1.txt*, October 1998.

[ea99a] B. Jamoussi et al. Constraint-based lsp using ldp. *Draft-ietf-mpls-cr-ldp-03.txt*, September 1999.

[ea99b] I. Stoica et. al. Per hop behaviours based on dynamic packet states. *internet draft. http://www.cs.cmu.edu/ istoica/DPS/draft.txt*, February 1999.

[For96] ATM Forum. P-nni specification, version 1.0. March 1996.

[IN98] J. Ibanez and K. Nichols. *internet draft : draft-ibanez-diffserv-assured-eval-00.txt*, August 1998.

[Kum74] K. Kummerle. Multiplexer performance for integrated line and packet switched traffic. *Int. Conf. Comput. Commun. Record*, pages 507–515, August 1974.

[Lee95] W. C. Lee. Topology aggregation for hierarchical routing in atm networks. *Comput. Commun. Rev. (USA)*, 25(2):82–92, 1995.

[ML97] H. Michiel and K. Laevens. Teletraffic engineering in a broad-band era. *Proc. of IEEE*, 85(12):2007–2033, December 1997.

[MMR96] D. Mitra, J. A. Morrison, and K. G. Ramakrishnan. Atm network design and optimization: A multirate loss network framework. *Proc. IEEE INFOCOM'96*, pages 994–1003, 1996.

[NBS95] A. Nguyen, N. Bambos, and M.H. Sherif. Adaptive (t1, t2)-multiplexing transmission schemes for voice/data integrated networks. *IEEE Syp. Comp. and Communications*, pages 430–435, 1995.

[NJZ97] K. Nichols, V. Jacobson, and L. Zhang. A two bit differentiated services architecture for the internet. *Internet Draft, draft-nicolas-diff-svc-arch-00.txt*, November 1997.

[OY94] Y. Ofek and M. Yung. The integrated MetaNet architecture: A switch-based multimedia LAN for parallel computing and real-time traffic. *IEEE INFOCOM'94*, 1994.

[RVC98] E. C. Rosen, A. Viswanathan, and R. Callon. Multi protocol label switching. *draft-ietf-mpls-arch-02.txt*, July 1998.

[SPG97] S. Shenker, C. Partridge, and R. Guerin. Specification of guaranteed quality of service. *rfc2211*, September 1997.

[SSZ98] I. Stoica, S. Shenker, and H. Zhang. Core-stateless fair queueing: A scalable architecture to approximate fair bandwidth allocations in high-speed networks. *Proc. of ACM SIGCOMM'98*, August 1998.

[SW97] S. Shenker and J. Wroclawski. General characterization parameters for integrated services network elements. *rfc2215*, September 1997.

[SZ98] I. Stoica and H. Zhang. Lira: An approach for service differentiation in the internet. *Proc. of NOSSDAV'98, Cambridge, England*, 1998.

[Wro97] J. Wroclawski. Specificiation of the controlled-load network element service. *rfc2211*, September 1997.

[Zaf74] P. Zafiropolo. Flexible multiplexing for networks supporting line switched and packet switched data traffic. *Int. Conf. Comput. Commun. Record*, pages 517–523, August 1974.

Integrating Active Networking and Commercial-Grade Routing Platforms

R. Jaeger[1,2], S. Bhattacharjee[1], J. K. Hollingsworth[1], R. Duncan[2], T. Lavian[2], F. Travostino[2]
[1]*University of Maryland, College Park, MD 20742*
[2]*Nortel Networks, 4401 Great America Parkway, Santa Clara, CA 95052*
{rfj,bobby,hollings}@cs.umd.edu {rduncan,tlavian,travos}@nortelnetworks.com

Abstract

Current network nodes enable connectivity between end-systems by supporting a static and well-defined set of protocols. The forwarding service provided by these network nodes is fixed, simple, and increasingly being implemented in hardware. Active network nodes, on the other hand, enable the unattended, dynamic instantiation of custom programs into the network node, allowing for the introduction of new protocols and services at runtime. Current prototype implementations of active network nodes achieve this flexibility by injecting a significant amount of software into the forwarding path.

This paper describes an Active Network platform that is ideally suited for integration into modern, commercial-grade network nodes, such as router and switches with silicon-based forwarding paths. This Active Network platform supports the dynamic introduction of application services that can alter packet processing; it comprises the Oplet Runtime Environment (ORE) and the Java Forwarding (JFWD) API. The ORE is the substrate that provides for the secure downloading, installation, and safe execution of network services. The JFWD API is a uniform, platform-independent portal through which software services can control the forwarding path of heterogeneous network nodes. We describe how existing active networking environments can be ported onto this Active Network platform and present performance results for dynamically loaded network services on the Accelar Gigabit Ethernet Routing Switch product.

***Index terms*-- Active Networks, distributed applications, networking protocols, NodeOS, ORE, Programmable Networks, JFWD**

1. INTRODUCTION

Traditional network nodes (e.g. routers on the Internet) enable end-system connectivity and sharing of network resources by supporting a static and well-defined set of protocols. The "virtual machine" defines the service provided by the network to transient traffic at each router; the customization of this machine is strictly limited to the configuration hooks that were envisioned at design time. The trend in commercial-grade routers and switches has been to implement ever more functionality of this virtual machine in hardware; hardware implementations have, in turn, enabled ever faster realizations of network nodes. However, the gain in raw performance due to hardware implementations is—almost by necessity—paired with a loss of customization options supported by the router on the data path. As more of the router's virtual machine is *frozen* in silicon, less are the opportunities to introduce new services inside the network.

1.1 Flexible Forwarding: Active Networks

Active Networks (AN) blurs the dichotomy between transient data packets and strictly node-resident software. Unlike traditional networks, AN enable the introduction of network services "on-the-fly". For instance, AN support per-flow customization of the services provided by a network node, according to the various notions of flow being used. The tenet of active networking is as follows: the utility of the service rendered by the network to individual applications is maximized if applications themselves are given the opportunity to define this service. In their most elaborate form, AN introduce a Turing-complete virtual machine at each router. Network users inject a "program" along with their data into the network. This program defines exactly how the network should process a user's data. Depending on the definition of user and the granularity of customization supported by the network interface, the network service in an active network may even be customized on a per-packet basis.

Obviously, such a broad definition of services supported by the network make the implementations of active network nodes difficult. Depending on the user-network interface, the active network implementation—almost by necessity—has to incorporate a substantial software component into the

data path [16].[1] Implementations of traditional and active networks must confront the age-old tradeoff between performance and flexibility. In this paper, we explore one point in the performance-flexibility space: we describe an implementation of active network techniques on a commercial routing platform.

1.2 Active networking applied to commercial hardware

This work captures the experiences and lessons-learned while porting our AN platform to the Nortel Networks Accelar Gigabit Routing-Switch. The primary goal of our work is to build a working platform for implementing programmable services on a commercial-grade, best-of-breed router. In doing so, we have tried to (a) preserve the router hardware fast-path for data packets, and (b) leverage existing active networking research as much as possible.

Obviously, (a) implies that certain computations that require data plane flexibility are not possible in our implementation. A paramount goal of our work is to identify sets of computations that become possible as additional functionality is placed into hardware. As part of our work, we also identify the broad "classes" of computations that are excluded by our implementation.

To support goal (b), we have implemented a layer over which existing active network implementations can be ported. In active networking parlance, this work does not introduce a new AN execution environment (EE). Rather, we present a Java-based run-time environment (the Oplet Run-time Environment) for security and service management over which existing EEs can be deployed and executed as network services. There is an important point of departure in our approach compared to the current work in the active network NodeOS community: we do not simply *trust* (even) Java-based EEs to conform to the resource limit policies set by the node provider. Instead, the ORE checks and enforces these limits on a per-EE basis at run-time. Further, the ORE can, as a matter of node policy, revoke resources and privileges granted to an EE during its execution. We have ported the ANTS EE to run within the ORE on the Accelar router. Our implementation architecture could allow porting the entire DARPA NodeOS interface onto our platform and support both Java-based and "native" EEs.

[1] In cases of both traditional and active networks, it may be possible to provide fast and customizable forwarding by incorporating hardware that is both programmable over a relatively fine time-scale and is able to forward packets at line-speeds (e.g. fast and programmable FPGAs).

1.3 Roadmap

In Section 2 we provide a brief overview of the DARPA active network architecture, followed by a description of the internal architecture of the Accelar router. We discuss the issues that must be resolved before the DARPA AN architecture can be realized on a commercial-grade, hardware-intensive routing platform and describe our mapping of the DARPA AN architecture to the Nortel Networks Accelar Gigabit Routing-Switch. In Section 3 we describe how we realize selected portions of the AN architecture on the Accelar, provide details of each component of our implementation, and describe the interfaces supported by each layer. In Section 4 we present a set of performance results from our implementation. In Section 5 we present related work and compare our implementation with existing work both in an architectural context and with respect to supported in-network computations. We present conclusions in Section 6.

2 BACKGROUND

In this section, we provide a quick synopsis of the DARPA active networking architecture and of the internal architecture of the Accelar platform.

2.1 DARPA Active Network Architecture

Figure 1 shows the node architecture for active networks developed by the DARPA active network research community [1].

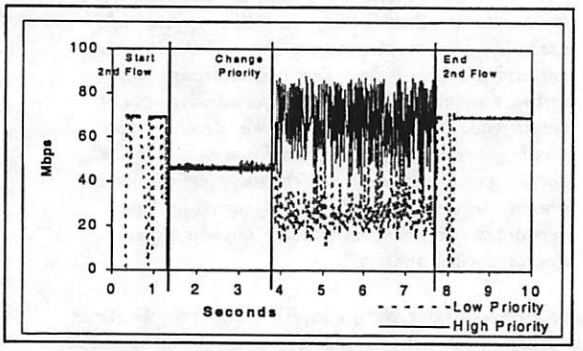

Figure 1: DARPA Active Network Architecture.

The DARPA active network architecture defines a framework through which APIs are exposed to applications by the active network node. The architecture must entertain more than one type of API; the so-called *Execution Environments* (EEs) have the mission to realize the various, specific APIs. The type of APIs and the number of EEs are not necessarily

known a priori. This implies that the architecture must define the "virtual machine" supported by network nodes in the network, and the extensibility paradigms associated with such a virtual machine.

EEs can implement a wide range of APIs that exploit different points in the trade-off between performance and flexibility, e.g. IPv4 can be considered a high performance EE that does not provide much flexibility while the ANTS [16] is an EE that provides a Java virtual machine at each node and sacrifices some performance for enhanced flexibility. By supporting multiple EEs, the architecture allows the user of the network to make an application-specific choice in this spectrum between performance and flexibility.

The native computation, communication, and storage resources at an active node are controlled by a *Node Operating System* (Node OS). The node OS provides an interface that exposes the resources available at the active node and mediates access to these resources. The node OS demultiplexes incoming packets to specific EE(s); EEs specify the subset of packets that must be handed-off to them. The node OS also provides support for *common objects* such as routing tables that are likely to be shared across EEs.

2.2 Nortel Accelar Router

The Nortel Networks Accelar family of L3 Routing Switches employs a distributed ASIC-based (Application Specific Integrated Circuit) forwarding architecture with a 5.6-256 Gbps per second backplane. Each ASIC is responsible for four physical 10/100 Ethernet ports or a single gigabit port. The switches scale up to 384 10/100 ports or 96 Gigabit ports (or some combination of the above). There are up to eight hardware-forwarding queues per port corresponding to normal and high priority packets. The hardware is controlled using the VxWorks real-time OS.

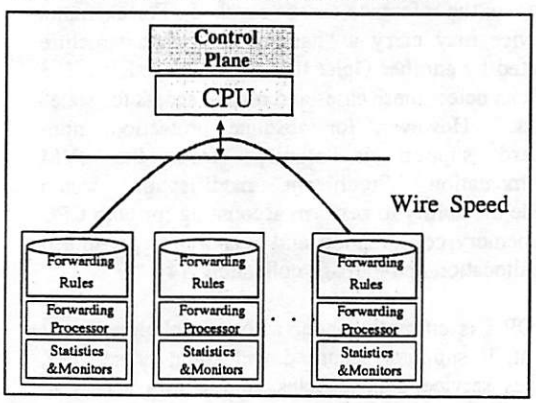

Figure 2: Architecture of the Accelar Router

Native applications monitor and control the ASIC hardware via a switch-specific API. This API provides access to hardware instrumentation variables to give native applications a one-to-one mapping to hardware functions. For example, the switch hardware provides functionality to set certain bits on an IP packet header that match a specific filter. This functionality is driven through a switch-specific API. Through this API, native applications can install packet filters that can inspect and modify packet headers at wire-speed.

3 Implementation

In this section, we describe how we realize selected portions of the AN architecture on the Accelar platform. In order to transform the Accelar routing switch to be a programmable network service platform, we implemented a run-time environment over which existing active network EEs can be executed. In general, this would require the implementation of the active network NodeOS API over the Accelar embedded real-time OS. However, the AN NodeOS API [2] was still evolving when we started our work and most EEs are implemented either within a JVM[16, 17] or over legacy OS interfaces. Thus, the path we chose was not to port/implement the NodeOS API and to limit support to Java-based EEs. There are two required steps in order for Java-based EEs to execute on the Accelar: (1) a JVM must be ported to the Accelar, and (2) a Java-compatible interface must be provided to the low-level hardware. Figure 2 shows a schematic diagram of the different components of our approach.

The embedded Java VM required by step (1) is a fairly well understood engineering task. JVMs can easily be ported to run as one of VxWorks' "tasks". The service degradation due to a single (possibly malfunctioning or malicious) JVM task on VxWorks is constrained. This is because the JVM runs as just another task in the real-time VxWorks O/S with a fixed and upper-bounded processor share and priority.

Step (2) required us to define a Java API to access low-level forwarding paths like the ones found on the Accelar. As forwarding paths can be heterogeneous—e.g., they can be implemented in software, ASICs, or network processors, and can have vastly different feature sets—it was crucial to come up with a forwarding API of wide applicability—i.e., not a point solution for the Accelar. The details of the resulting Java Forwarding API (JFWD) are captured in Section 3.3.

Though not technically a necessity, we added separate layer between the JVM and the EE. This layer —the Oplet Runtime Environment (ORE)—provides security and management services that may eventually be subsumed by the AN NodeOS and was deemed to be a necessity for the commercial viability of the activated routers. As mentioned before, the ORE enables a stricter intra-node trust infrastructure allowing for different per-node resource allocation policies without cooperation from EE writers. Thus, ORE provides mechanisms for nodes to enforce per-EE resource limits without having to trust the EE. A nice corollary is that the ORE allows multiple EEs (or multiple instantiations of the same EE) to be spawned within a single Accelar with different privileges. In the next section, we present details of the ORE and JFWD API.

3.1 ORE: The Oplet Run-time Environment

The ORE is a platform for secure downloading, installation, and safe execution of Java code (called *services*) within a JVM. A service is a monolithic piece of code that implements specific functionality. A service may *depend* on other services in order to execute. In order to securely download and impose policy, we introduce the notion of "Oplet". Oplets are self-contained downloadable units that embody a non-empty set of services. Along with the service code, an Oplet specifies service attributes, authentication information, and resource requirements. Note that Oplets can encapsulate a service that depends on some other service; in these cases, Oplets also contain dependency information. In general, the ORE must resolve and download the transitive closure of Oplet dependencies before executing a single service.

The ORE provides mechanisms to download Oplets, resolve dependencies, manage the Oplet lifecycle, and maintain a registry of active services. The ORE uses a public-key infrastructure to download "trusted" Oplets. In brief, the security infrastructure provides authentication, integrity, and non-repudiation guarantees on downloaded Oplets. Due to space restrictions, we will not elaborate more on the secure downloading, execution, or resource management features of Oplets.

3.2 Oplet Execution Safety

The ORE must provide safe execution and impose resource limits. As far as possible, the ORE uses the mechanisms provided by the Java language (type safety) and the standard JVM (bytecode verification, sandbox, security manager) to provide execution safety. The ORE controls allocation of system resources by intercepting allocation calls from the service code to the JVM.

To protect itself from denial of service attacks, deadlocks, and unstable states, the ORE implements mechanisms for thread safety and revocation. The ORE controls thread creation by requiring Oplets to request new threads from the ORE. The ORE determines whether to grant the request based upon a node policy that takes into account current thread usage, and the credentials of the requesting Oplet. Once a thread is allocated, however, the current implementation of the ORE has no mechanism in place to account for or limit the consumption of computing resources. In its most general form, the ORE must address denial of service caused by a misbehaving service unduly consuming CPU resources. To handle these issues, the ORE needs JVM support for CPU accounting [14].

Sharing threads between Oplets presents two main problems: (a) deadlock caused by a callee not returning and (b) caller Oplet killing the shared thread while it is executing in a callee Oplet's critical section. The ORE protects itself from the first problem by never interacting directly with any Oplet that it loads. Instead it creates a trusted proxy which the ORE uses to delegate its commands to the untrusted Oplet. The proxy uses a separate thread to call a method on the untrusted Oplet and sets a timeout for returning from the call; if the thread call does not return after a conservatively set timeout, a fail-stop situation is assumed and the thread is killed. The second problem is handled by the ORE by revoking Oplet's ability to manipulate a thread's running status.

The ORE uses object revocation to control access to its own resources. If the ORE determines that a specific service is no longer permitted to use a resource reference, the reference can be revoked. For example, a service may carry a "handle" to a data structure exported by another Oplet that no longer exists. The ORE can detect these cases and revoke access to "stale" objects. However, for absolute protection, non-standard support is required from the JVM implementation. Significant modification would include the ability to perform accounting for both CPU and memory consumption and support for per-thread heap allocation and garbage collection [14].

The ORE is currently under active development. At present, it supports secure downloading of services, resolves service dependencies, and allows access to native router functionality through the JFWD API. However, the current ORE version is still vulnerable

to several flavors of denial-of-service attacks. These include spurious triggering of the Java garbage collector, memory fragmentation attacks, and stalling finalization of objects[14]. Several memory related safety hazards confronting the ORE will be resolved as JVMs support multiple heaps, revocation and copy semantics of the JKernel [8].

3.3 JFWD: The Java Forwarding API

The JFWD API specifies a platform-independent interface for Java applications to control a virtual forwarding path of commercial-grade strength. The platform-independent nature of JFWD rests upon a) an extensible behavioral model of the forwarding path, and b) an extensible data model of control data (e.g., routing tables) that need to be fed into a forwarding path to affect its behavior. To port JFWD to any given platform, an engineer has to contrast the features of the target forwarding path with the ones modeled in the JFWD API specification, and then proceed to either pruning JFWD classes that are not applicable to the target forwarding path, or sub-classing existing JFWD classes to cope with platform-specific forwarding idiosyncrasies. Subsequent ports of the JFWD contribute feedback and new classes back into the JFWD API specification and its models, and this way the JFWD API evolves towards new forwarding technologies.

A selected set of JFWD classes has been ported to the Accelar. The implementation of these JFWD classes is highly platform dependent; on the Accelar, the JFWD classes turn out to be a wrapper around the hardware instrumentation interface. In the rest of this section, we highlight the main mechanisms that are provided by the JFWD API on the Accelar switch.

Among other things, the JFWD API can be used to instruct the forwarding path to alter packet processing through the installation of hardware filters. The hardware filters execute "actions" specified by a filter policy. On the Accelar, the filter can be based on combinations of fields in the MAC, IP, and transport headers. The policy can define where the matching packets are delivered and can also be used to alter the packet content. Packet delivery options include discarding matching packets (or conversely, forwarding matching packets if the default behavior was to drop them) and diverting matching packets to the control plane. Diverting packets to the control plane allows applications, such as AN EEs to process packets. Additionally, packets can be "carbon copied" to the control plane or to a mirrored interface. Packets may also be identified as being part of high priority flows; these packets can be placed in a static hardware high priority queue.

The filter policy can also cause packet and header content to be selectively altered (e.g. the Type of Service bits on matching packets can be set). The existing hardware is capable of re-computing IP header checksum information at line speeds even if the IP header has to be altered.

3.4 Network services supported by the JFWD and the ORE

In this section, we explore the set of possible and precluded computations on the platform defined by ORE and JFWD API. Note that the ORE does not, a-priori, exclude any computation; instead, it enforces node policy that may cause certain (e.g. processor-intensive) computations to not be started or terminated during execution. Computations are, instead, constrained by the JFWD API since this API defines those capabilities that are exported by the hardware and can be used to build network services.

Thus, some computations, e.g. certain video transcoding techniques that must process every packet, cannot efficiently be implemented in our system regardless of node policy. Not all precluded computations involve data transformation; certain network based anycasting/routing schemes in which a program must be executed to find the outgoing switch port cannot be supported either. The reason is that you are putting inherently slow computations into the forwarding process which is not sustainable at high data rates.

In general, all control-plane only computations, e.g. installing new routing tables or parsing a new ICMP message type, can be rather easily accommodated by the ORE/JFWD API. An important ability enabled by the JFWD API is to *selectively* route (or copy) packets to the control plane —as we will see, this does significantly enlarge the set of services that can be implemented on the Accelar. In the rest of this section, we identify a specific set (non exhaustive) of services that can be implemented using the current version of the JFWD API.

- **Filtering firewall** - One simple application would be a firewall that allows or denies packets to traverse on specified interfaces depending on whether the packet's header matches a given bit mask.

- **Application-specific firewall** - It is relatively straightforward to extend the filtering firewall implement certain application-specific firewalls. For example, an FTP gateway that dynamically changes the firewall rules to allow *ftp-data* connections to a ``trusted" host can be implemented. Security functions like stopping TCP segments with no (or all) bits set can also be dynamically programmed on the Accelar. Almost all modern routers allow for a filtering firewall and application-specific firewall functionality. On the Accelar ORE/JFWD platform, it is imperative to note that these services can now be added, modified, and deleted dynamically, on demand, and without human intervention. The next three services are example of features that, in general, are not yet available in most commercial routers.

- **Dynamic RTP flow identification** - RTP over UDP flows are identified by an ephemeral UDP port number. In general, some host chooses this port number and it is not well known. We have implemented several mechanisms to identify RTP flows traversing the Accelar. Using the JFWD API, control protocol (SIP/RTSP/H.323) messages can be intercepted and parsed for RTP port numbers. We are currently implementing a more dynamic solution that samples packets on specified interfaces and uses probabilistic techniques to identify/mark RTP flows.

- **DiffServ: Classifier, Marker** - The Accelar can be turned into a DiffServ[6] Classifier by suitably programming its hardware filters. Further, the hardware (and in turn, the JFWD API) provides mechanisms to change, at line-speed, selected bits in the IP header. This ability can be used to implement parts of DiffServ ingress/egress marker capabilities on the Accelar. A subtle benefit of this solution is that new firmware or hardware does not have to be shipped each time a new DiffServ scheme/PHB becomes popular. Instead, using existing ORE service instantiation mechanisms, only the service-specific logic has to be uploaded onto the router. This can be accomplished on-line, without interrupting existing flows or services.

- **PGM-like Reliable multicast** - The packet filtering capabilities of the Accelar allows certain packets to be copied on for inspection by the service code. This mechanism can be used to divert (negative) acknowledgements from multicast sessions to the control plane. The service code can, much like the PGM reliable multicast scheme, send one copy of the NAK upstream and suppress duplicate NAKs. Unlike PGM, modulo resource constrains, it is possible to implement reliable multicast services that keep a small packet cache and immediately re-transmit a lost segment. Other services, such as multicast ancestor discovery, can also be efficiently implemented by providing the service code interfaces to the routing table.

We conclude this section with a "wish-list" of a set of functions that, if implemented in hardware and exported by the JFWD API, would enable a new breed of network services. This wish-list is not meant to pinpoint shortcomings of the particular commercial platform that we have used and that is quite good at delivering the services that a traditional customer basis demands. To the contrary, the wish-list represents a constructive hint to those engineering teams endeavoring on new projects explicitly aimed to programmable, active network nodes.

There are two types of functions that are required: functions that are "better" substitutes for existing functionality, and functions that are not available, in any form, in the existing implementations.

There are two main elements in the first class of functionalities with marginal improvement. The first one is the replacement of static priority output scheduler with a better scheduling algorithm (e.g. weighted fair queuing). This would enable RSVP[5] functionality to be implemented as a service. The second one is the ability to discard frames with a given probability function. To implement RED[7] and its variants, a primitive of this kind needs to be added to the discard/divert/forward/copy semantics of permissible actions upon hardware filter match.

We conclude this section with two useful functions that do not exist, in any form, in the Accelar hardware.

- **Token Bucket** - The Accelar hardware could be augmented to provide support for a set number of token buckets, each with a configurable buffer and draining at a specified rate. Obviously, DiffServ shapers and assorted RSVP policy can be implemented using this mechanism.

- **Queue Exposure and manipulation** - The Accelar hardware/JFWD API does not provide any mechanisms for services to get a "sample" or snapshot of the set of currently queued packets. Application-specific congestion control functionality [9] can be implemented using an

interface that allows services to periodically check if packets of a certain type (i.e. matching a specified ALF header) from a given flow are queued on an output port. An extension to the queue exposure interface allows services to delete (in the general case, transform) packets that are already queued. The queue exposure and manipulation techniques have been applied to significantly improve end-to-end quality of media streams [4, 9].

4 Experimental Results

In this section, we describe a simple experiment on the Accelar platform. The experimental topology is shown in Figure 3. The hardware used in the experiment included a Accelar 1100B Routing Switch configured with 16 10/100 Mbps port and a 5.6Gbps backplane. The three hosts ran GNU/Linux (kernel version 2.2.5) over 233 Pentium II processors.

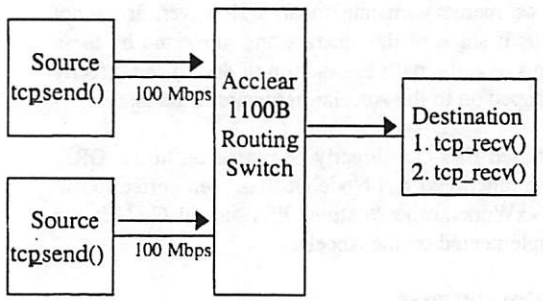

Figure 3: Experimental Setup.

During the experiment, we sent two TCP flows from the two sources to the single destination. In Figure 4 we show the results from a sample run of our experiment: the x-axis corresponds to absolute time at the receiver with respect to a clock that started when the first packet is received; the y-axis corresponds to measured bandwidth in application-space at the receiver averaged over 48 1200 byte segments. Note that for our purposes, the received clocks are synchronized as each process samples current time from the same hardware clock using the Unix `gettimeofday` library call. Once the second flow starts (at time 1.3 seconds), the source TCPs contend for bandwidth on the output link and stabilize their data rate at about 47 Mbps each. We then use an downloaded ORE service (at time 3.8 seconds) to dynamically increase the priority of the second flow. In this case, the service does not implement dynamic flow detection, instead it just uses a fixed source address based filter to discriminate packets from each of the sources. As expected, the received bandwidth on the second (high priority) flow increases and stabilizes at about 70 Mbps. On our testbed, this is the maximum end-to-end bandwidth attainable without any contention. After the second flow ends (at time 7.7 seconds), the low-priority TCP flow can increase its rate and increases its rate up to the expected 70 Mbps.

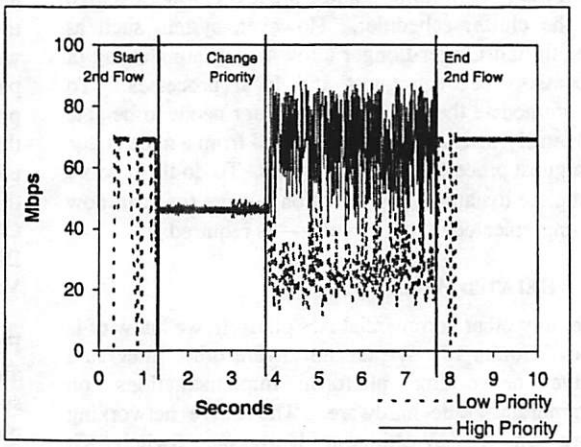

Figure 4: Experimental Results for Dynamic Assignment of Priority to Flows.

4.1 Discussion

In isolation, the experiment and the results described above do not qualify as new behaviors. The novelty, however, is derived from the fact that the priority assignment code was installed dynamically on a commercial-grade router capable of stably supporting a large workgroup. In this section, we discuss an immediate application of this functionality that we are using in our own research facilities.

An immediate benefit of on-line identification of flows and dynamic adjustment of packet priority is to support cluster computing. In cluster systems such as Condor[11], NOW[3], Stealth[10], and Linger-Longer[15] workstations are used to run jobs when the computer's primary user is not using their computers. To make these systems usable, the software that runs guest jobs on user's workstations goes to great lengths to ensure that the guest process does not interfere with the primary user. However, until now there has been no clean way to isolate guest use of a workstation from network traffic generated by normal users.

By using active networking at the local area network switch, we can dynamically identify the flows associated with guest jobs. Although these jobs typically have a set of well-known ports, they also can use other network services. To help identify these

flows, the cluster scheduler software, can inform the switch when a particular node has started to run a guest process. For some clusters such as Condor and NOW, a node in a cluster is either running guest processes or local process and switches between them on a time-scale measured in minutes. For these types of systems, a simple filter to re-prioritize all traffic from the host running a guest process can be installed by the cluster scheduler. However, system such as Stealth and Linger-Longer allow fine-grain sharing of processors between guest and local processes. To accommodate these systems, the filter needs to be able to identify whether traffic associated from a node is due to a guest process or a local process. To do this a more complete dynamic flow detection —one that can now be implemented on the Accelar— is required.

5 Related Work

The only other commercial AN platform we know of is from 3Comm[18]. We are not aware other integrated active networking platform implementations on commercial-grade hardware. The active networking work on the Washington University Switch Kit employs locally connected machines as active processors[2]. The Tempest [13] provides a customizable control plane for ATM networks. The basic ideas of high-performance active networking by decoupling the forwarding path from a programmable control plane was introduced, in a software implementation, in the Control-on-Demand (CoD) [9] platform co-developed at AT&T Labs. In this section, we compare our approach to CoD, and discuss how existing active networking research fits within our framework.

The Control-on-Demand platform was developed and implemented over IPv6 as an extension to the Linux kernel [9][3]. Data packets were kept in the kernel in per-flow queues while active control could be applied to the data packets by dynamically loaded "per-flow controllers" that executed in user space. The per-flow controllers affected the data path using the CoD API. A meta-controller loaded each per-flow controller using a signaling protocol. CoD was developed to be specifically mapped onto hardware platforms and its relationship to our work is clear. Services on the Accelar map to per-flow controllers in CoD; the JFWD API on the Accelar maps to the CoD API; the ORE functionality on the Accelar is not completely replicated in CoD, though the meta-controller does provide a subset of the ORE functionality. As CoD was implemented in software; it provides all of the JFWD functionality, and also provides the queue exposure and manipulation facilities on our hardware wish list.

Active networking NodeOS's can potentially be implemented over VxWorks on the Acclear. There is one fundamental problem: the AN NodeOS architecture allows for all packets on specific channels to be delivered to the EE for further processing —this would negate the benefits of the hardware forwarding path available in the Accelar. However, the Accelar provides a perfect platform for implementing fast cut-through paths. The Bowman NodeOS[12] is a particularly good fit as it is specifically supports cut-through paths and is designed as a layer above a host OS that provides low-level hardware access. Thus, Bowman can directly be ported on to the Accelar using VxWorks as its host OS.

For other Node OS efforts, the VxWorks platform already implements much of the required functionality such as memory management. However, it is not obvious if some of the abstractions supported by these systems (e.g. the path abstraction in Scout) can directly be mapped on to the Accelar hardware features.

Java-based EEs can directly be ported on to the ORE. Once a functional AN Node OS has been ported to run over VxWorks, other "native" EEs such as CANEs can be implemented on the Accelar.

6 Conclusions

We presented a summary of the challenges of bringing Active Networking ideas to current high performance hardware-based routers and switches. In addition, we showed that while it is not currently feasible to support active packets for every packet at line speed on these systems (nor any system), it is possible to exploit existing hardware filtering mechanisms to allow a variety of scenarios that require active functionality on routers. To demonstrate the feasibility of our approach, we presented results from an initial implementation of Active Networking support on the Nortel Accelar. This example showed that it is possible for existing hardware to be able to support active networking environments such as ANTS. Also, we have described how the programmable features of existing ASIC-based hardware forwarding engines can be used as a building block for extensible networks services.

[2] See http://www.ccrc.wustl.edu/gigabitkits/kits.html

[3] Control on Demand was co-developed by S. Bhattacharjee

7 REFERENCES

1. "Architectural Framework for Active Networks Version 0.9," . August 31, 1999,.Active Networks Working Group.
2. "NodeOS Interface Specification," . June 15, 1999,.AN Node OS Working Group.
3. R. H. Arpaci, A. C. Dusseau, A. M. Vahdat, L. T. Liu, T. E. Anderson, and D. A. Patterson, "The Interaction of Parallel and Sequential Workloads on a Network of Workstations," *SIGMETRICS*. May 1995, Ottawa, pp. 267-278.
4. S. Bhattacharjee, *Active Networks: Architectures, Composition, and Applications*,Ph.D., Computer Science Department Georgia Institute of Technology,1999.
5. R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin., *Resource ReSerVation Protocol (RSVP)*, RFC 2205, , September 1997.
6. D. Black, S. Blake, M. Carlson, E. Davies, Z. Wang, and W. Weiss, *An Architecture for Differentiated Services*, RFC2475, , Dec. 1998.
7. S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, **1**(4), 1993, pp. 397-413.
8. C. Hawblitzel, C. Chang, G. Czajkowski, D. Hu, and T. v. Eicken, "Implementing Multiple Protection Domains in Java," *USENIX Technical Conference Proceedings*. June 1998.
9. G. Hja'lmtysson and S. Bhattacharjee, "Control on Demand: An efficient approach to router programmability," . April 1999.
10. P. Kruger and R. Chawla, "The Stealth Distributed Scheduler," *ICDCS*. 1991, pp. 336-343.
11. M. Litzkow, M. Livny, and M. Mutka, "Condor - A Hunter of Idle Workstations," *International Conference on Distributed Computing Systems*. June 1988, pp. 104-111.
12. S. Merugu, S. Bhattacharjee, E. Zegura, and K. Calvert, "Bowman: A Node OS for Active Networks," *to appear INFOCOM'2000*.
13. J. E. v. d. Merwe, S. Rooney, M. Leslie, and S. A. Crosby, "The Tempest - A Practical Framework for Network Programmability," *IEEE Network*, **12**(3), 1998.
14. P.Bernadat, D. Lambright, and F. Travostino, "Towards a Resource-safe Java for Service-Guarantees in Uncooperative Environments," *IEEE Symposium on Programming Languages for Real-time Industrial Applications (PLRTIA)*. Dec. 98, Madrid, Spain.
15. K. D. Ryu and J. K. Hollingsworth, "Linger Longer: Fine-Grain Cycle Stealing for Networks of Workstations," *SC'98*. Nov. 1998, Orlando, ACM Press.
16. D. Wetherall and e. al., "ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols," *OPENARACH'98*. 1998.
17. Y. Yemini and S. da Silva, "Towards Programmable Networks," in IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, L'Aquila, Italy, October, 1996
18. D. Nessett, "Commercial use of Active Networking," OpenSIG Workshop, University of Toronto, October 5-6, 1998

RSIP: Address Sharing with End-to-End Security

Michael S. Borella
3Com Corp.
1800 W. Central Rd.
Mount Prospect, IL 60056
mike_borella@3com.com

Gabriel E. Montenegro
Sun Microsystems Laboratories
901 San Antonio Road, MS UMPK15-214
Mountain View, CA 94303
gab@sun.com

Abstract

Realm Specific IP (RSIP) is a new architecture under consideration in the Internet Engineering Task Force (IETF) that can potentially alleviate some of the problems associated with partitioning of the Internet address space due to, for example, the shortage of IPv4 addresses. It is being positioned as a replacement for Network Address Translation (NAT), because, among other things, it can support end-to-end security via IPsec, which NAT cannot. This paper introduces the motivation behind RSIP, the RSIP architecture, and provides a basic overview of the RSIP protocol.

1 Introduction

IPv4, the current version of the Internet Protocol, supports 32 bits of address space, which means that over 4 billion individually addressable hosts can be on the Internet. At the time of its inception, IPv4 designers could not have imagined the explosive growth of the Internet in the mid-to-late 1990's. As a result of this growth, as well as overly generous address allocation schemes of the past, it is becoming increasingly difficult and expensive to obtain IP addresses. Although IPv6, which has 128 bits of address space, has been approved by the Internet Engineering Task Force (IETF), there is no clear upgrade path from IPv4 to IPv6. Concerns that IPv6 may either never deploy, or may only partially deploy, are being expressed [1].

Thus far, network users and administrators have responded to the address shortage by deploying Network Address Translation (NAT) [2] in boundary routers. However, this technology requires that the NAT be aware of any application run across it that transmits IP addresses or TCP/UDP port information in the packet payload. Furthermore, NAT inhibits the use of end-to-end security via IPsec [3]. The latter, in particular, is a major disadvantage, considering the popularity of virtual private networks and the well-known need for security in e-commerce and business communications.

Currently, network administrators must choose between a complete solution that is not supported by many vendors and incompatible with the rest of the Internet (IPv6), and a stop-gap solution that introduces many problems and prevents the deployment of some popular applications (NAT). Realm Specific IP (RSIP) [4] has been proposed as a replacement for NAT that will have little impact on application layer protocols, and will inter-operate with IPsec. RSIP has the additional advantage that it can coexist with NAT, and therefore networks using NAT can be smoothly upgraded, in part or in whole, to use RSIP.

This paper presents an introduction to RSIP, focusing on how it is different from NAT, and how it can support end-to-end IPsec. Since RSIP is still in the draft phase of its journey towards IETF standardization, we include a discussion of architectural and protocol issues that are currently being addressed and evaluated.

2 Background

In this section we discuss the IP address shortage and how it is expected to become a critical issue in the near future. We then describe the operation of NAT, which is required in order to appreciate the alternative

2.1 The Coming IP Address Crunch

The technical media and popular press have been creating much hype about the IP address shortage. Currently, it is still possible to obtain IP addresses in most parts of the world. However, a requester typically has to justify that they actually need the number of addresses that they request, and must demonstrate use of the addresses that they are allocated. Furthermore, addresses have become a commodity of sorts, as ISPs typically charge significantly more for additional addresses beyond the default number that they give to a client.

In the near future, we expect that three trends will cause the rapid exhaustion of the remaining IPv4 address space.

1. *The Internet revolution reaches Asia and developing nations*: Over one third of the world's population currently resides in China and India, two countries that are relatively poorly connected. Given the growing availability of inexpensive access technologies, as well as wireless networking, we can expect a sharp spike in address demand once these countries come online en masse.

2. *Home networking takes off*: All indicators are suggesting that residential networks will be commonplace within the next 2-5 years. The number of multiple PC households is growing, and it is expected that network appliances, from thermostats to physical security systems to microwave ovens, will follow. Potentially, the well-wired household may require dozens of addresses, and a multiple dwelling unit may require hundreds.

3. *Proliferation of cellular IP phones and PDAs*: Several major cellular telephony manufacturers have recently announced that they intend to deploy IP-aware "smart phones," that will replace both current cellular phones and personal digital assistants (PDAs). If this occurs, there will be a sudden worldwide need for millions, if not billions, of addresses.

These trends indicate that within 12-36 months we will begin to feel the IP address crunch in earnest, and that not long thereafter, it may become severe.

2.2 Network Address Translation

The address shortage began to be felt in the early-to-mid 1990's. NAT was developed as an intermediate, temporary solution, that would hold us over until IPv6 deployed. Obviously, IPv6 has not deployed, and even IPv6 advocates admit that unless there is a clearly defined transition phase, it may never deploy. As a result, NATs are becoming widespread, especially in the small enterprise and home networking space.

In principle, a NAT is a boundary router between two different address spaces. Typically, one is a private space of an ISP, residential network, or corporate intranet, and the other is the public space of the Internet. The hosts in the private space use private IP addresses [5] that are unroutable from the public Internet. The NAT performs a one-to-one translation of outgoing (private-to-public) packets from each private address to a unique public IP address, and performs the converse operation for incoming (public-to-private) packets. A popular variation of NAT, Network Address and Port Translation (NAPT) maps all private addresses to one or more public addresses, differentiating amongst these hosts by local port number. Thus, a NAPT device must ensure that all port numbers used on the local side of a session are unique per public address. This requires that some port numbers chosen by a host will be translated along with their address.

The major drawback that NAT introduces is that if an application transmits IP addresses or ports as part of its packets' payloads, the NAT must contain an application layer gateway (ALG) in order for it to support the protocol. The classic example of this is FTP. In the FTP control stream, the client transmits the IP address and port number to which the server should open a socket. In order for FTP to work across a NAT, the NAT must examine the payload of FTP control packets, determine where the address and port information is encoded, and perform the necessary translation. In the worst case, this requires that the NAT also modify the packet length and sequence number fields in the IP and TCP headers, respectively, TCP header checksum, and maintain a running delta of the TCP sequence number for lifetime of the connection.

The need for a protocol-specific ALG in the NAT for each protocol that transmits address or port content is more of a deployment issue than an engineering is-

sue. As long as a protocol payload can be decoded, read, and modified without disrupting end-to-end communications, NAT manufacturers can develop an appropriate ALG. However, deploying this ALG to an installed base of customers can prove to be trying. Since new protocols are being developed at a record pace, a NAT user must perform software or firmware upgrades on a regular and frequent basis. Despite these limitations, the utility of NATs has so far outweighed these drawbacks. Perhaps the true showstoppers for NAT, however, are applications that cryptographically prevent NATs from modifying IP packets. In general, this rules out end-to-end application of IPsec. In particular, NAT breaks all applications of the authentication header (AH), and applications of the Encapsulating Security Payload (ESP) in so-called transport mode. ESP in tunnel mode, however, is impervious to modifications of the outermost IP header. In spite of its deleterious effect on IPsec, NATs do not completely hinder the use of end-to-end data security. Security mechanisms at or above the transport layer, such as TLS or SSH, are unaffected if the applications being run do not transmit addresses or ports in their payloads.

Nevertheless, given the increasing demand for end-to-end network layer security, typically in the form of the virtual private networks that IPsec enables, NAT is seen as a critical roadblock for these services.

3 RSIP

RSIP is an alternative to NAT that operates under the same assumptions of physical architecture and connectivity. While NAT is only defined in terms of operations on a flow of packets, RSIP is defined in terms of operations on the flow, and also a signaling association between the client host (RSIP client) and gateway router (RSIP server). The nature of the operations on the flow of packets is also quite different. NAT gateways must match incoming flows with arbitrary filters. The filters used by RSIP servers use only a very small fixed number of prefixes, which lends itself much more to efficient implementation in hardware.

3.1 Locating RSIP Gateways

Before a client can invoke the services of an RSIP server, it must first locate the RSIP server. That is, it must obtain the RSIP server's IP address. Obviously, this information can be manually configured in each client, but it is highly desirable to automate the discovery process, especially in situations in which a roaming client is visiting a "foreign" network.

Routers between a client and the RSIP server will not intercept and relay RSIP messages like is common for DHCP. Since on most networks, clients will obtain their local addresses with DHCP, the IETF currently is defining a DHCP option that informs clients of the address of the RSIP server [6]. An alternative that is also under consideration is to use the Service Location Protocol (SLP) [7]. This protocol provides a framework for highly dynamic query-based discovery of services. SLP clients (user agents) can discover only those services (service agents) that satisfy certain criteria expressed by the client. Services are defined by the SLP templates. There is now such a template for SLP-based discovery of RSIP servers [8].

3.2 Packet Flow

The key to RSIP is for the RSIP client to prepare packets that are ready for the public network, such that no translation is necessary by the RSIP server. In order to do so, the RSIP client queries the RSIP server for the appropriate public address and port number(s) to utilize (see Section 3.3 for details), and then prepares a packet using these parameters. The RSIP client tunnels this packet over the private network to the RSIP server, which then only has to strip off the tunnel header, and forward the packet on to the public network.

An example RSIP packet flow is shown in Figure 1. An RSIP client (address 10.0.0.4) is connected to a multi-homed RSIP server by a private network. The RSIP server maintains one interface on the private network (address 10.0.0.1) and one interface on the public network (address 149.112.240.55). The figure illustrates a typical HTTP request-reply transaction between the RSIP client and a public WWW server (address 192.156.136.22). It is assumed that the RSIP server has allocated its public IP address

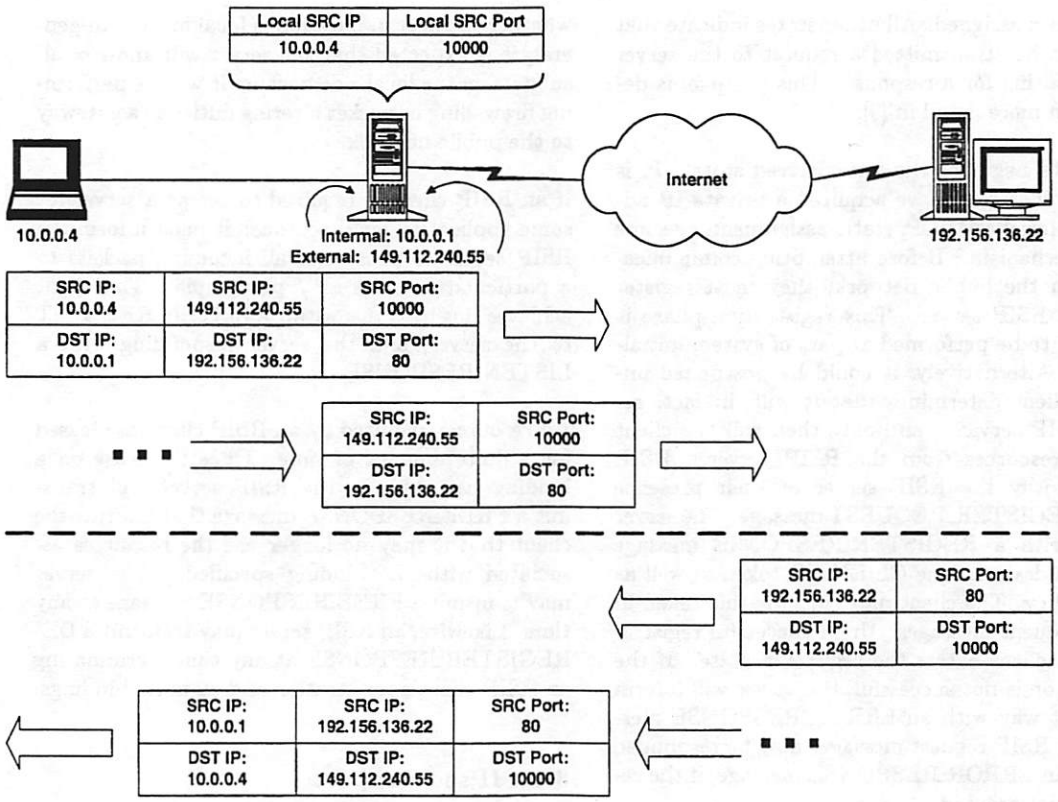

Figure 1: RSIP media flow.

and port 10000 to the RSIP client. When creating the HTTP request packet, the RSIP server uses port 10000 as the source port and 149.112.240.55 as the source IP address. This packet is then encapsulated in a tunnel that delivers it across the local network, to the RSIP server. The RSIP server removes the tunnel header and forwards the packet on the public network. For the incoming response from the WWW server, the RSIP server performs a lookup on the destination port number. Finding port 10000 associated with address 10.0.0.4, the RSIP server forms the tunnel header and transmits the encapsulated packet to the RSIP client.

Since the RSIP client prepares the packet to appear as if it originated from the RSIP server, there is no need for an ALG. Furthermore, even if the RSIP client is using end-to-end network layer encryption with a public server, the transaction will operate properly through the RSIP server, because the RSIP server does not need to examine the payload contents. In general, the RSIP server will allocate more than one port number per RSIP client; thus, the client can utilize protocols, such as FTP, that require multiple simultaneous sessions.

When an RSIP client communicates on the private network, it uses its local (private) address, and is not restricted by RSIP in any way. Thus, an RSIP client must respond to ARPs for its private address, but it must not respond to ARPs for a public address that it is using.

3.3 Signaling

Before an RSIP client can contact a public host, it must establish a signaling association with the RSIP server. The association can be either a TCP connection or a UDP session. It allows the RSIP server to lease address and port bindings to the client, de-allocate these bindings, and otherwise manage resources. The RSIP protocol runs in a simple request-response format. There are three major states that a client may be in: unregistered, reg-

istered, and assigned. All other states indicate that the client has transmitted a request to the server and is waiting for a response. This protocol is described in more detail in [9].

All clients begin in the unregistered state. It is assumed that they have acquired a private IP address, either via DHCP, static assignment, or some other mechanism. Before attempting communication with the public network, they must register with the RSIP server. This registration phase is expected to be performed as part of system initialization. Alternatively, it could be postponed until the client determines that it will, in fact, require RSIP services, and only then will the client request resources from the RSIP server. RSIP clients notify the RSIP server of their presence with a REGISTER_REQUEST message. The server replies with a REGISTER_RESPONSE message that includes a unique CLIENTID token as well as other policy. The client must include this token in all subsequent messages. Upon successful registration, the client enters the registered state. If the registration is not successful, the server will inform the client why with an ERROR_RESPONSE message. All RSIP request messages may be responded to with an ERROR_RESPONSE message, if the request is not granted.

Once registered, a client may request a public IP address and one or more ports with an ASSIGN_REQUEST message. Once an associated ASSIGN_RESPONSE is received from the server, the client enters the assigned state. Each ASSIGN_RESPONSE includes a per-client unique BINDID that identifies the bound resources. In the assigned state, the client may communicate with public hosts, request more resources with another ASSIGN_REQUEST message, free some assigned resources with a FREE_REQUEST message, or de-register with a DE-REGISTER_REQUEST message. From the assigned state, a de-registration frees all resources bound to the client.

Although RSIP is initially targeted at home networks and small to medium enterprises, it may also be deployed in large enterprise networks. These networks may include hundreds of subnets behind an RSIP server, and may require the RSIP client to know whether a given host is on the local or remote side of its RSIP server. To facilitate this situation, a client may transmit a QUERY_REQUEST to the server with the address of a host. The QUERY_RESPONSE from the server will indicate whether the host in question is local or not. In general, it is expected that the server will know of all subnets on the local side because it will be performing firewalling or packet filtering duties as a gateway to the public network.

If an RSIP client is required to act as a server for some application layer protocol, it must inform the RSIP server to pass to it all incoming packets to a particular IP address / port tuple. The client achieves this by transmitting a LISTEN_REQUEST to the server, and the server responding with a LISTEN_RESPONSE.

All resources acquired by an RSIP client are leased for a finite amount of time. Once the lease on a binding has expired, the RSIP server will transmit a FREE_RESPONSE message that informs the client that it may no longer use the resources associated with the binding specified. The server may transmit a FREE_RESPONSE message at any time. Likewise, an RSIP server may transmit a DE-REGISTER_RESPONSE at any time, terminating an RSIP client's registration and all of its bindings.

3.4 IPsec Support

IPsec enables secure end-to-end communication. Packets can either be encrypted, authenticated, or both. In order to use IPsec, two hosts must first establish a security association (SA), perhaps using the Internet Key Exchange (IKE) [10] protocol. All packets protected by an SA have at least one extra header inserted between the IP header and the transport layer header. Figure 2 shows IPsec packets in transport mode. The other of two possible modes is tunnel mode, in which the AH or ESP headers are followed by another IP header. The encapsulating security payload (ESP) header, shown in Figure 2a, encrypts the entire packet payload, and optionally authenticates the entire packet payload except for part of the ESP trailer. The authentication header (AH), shown in Figure 2b, authenticates the entire packet including the immediately preceding IP header (except, of course, for the IP header fields that change per hop). Both ESP and AH may be applied at a packet, as is shown in Figure 2c.

Even though ESP does not include the preceding IP header in its cryptographic calculation, it does include the entire payload, which in transport

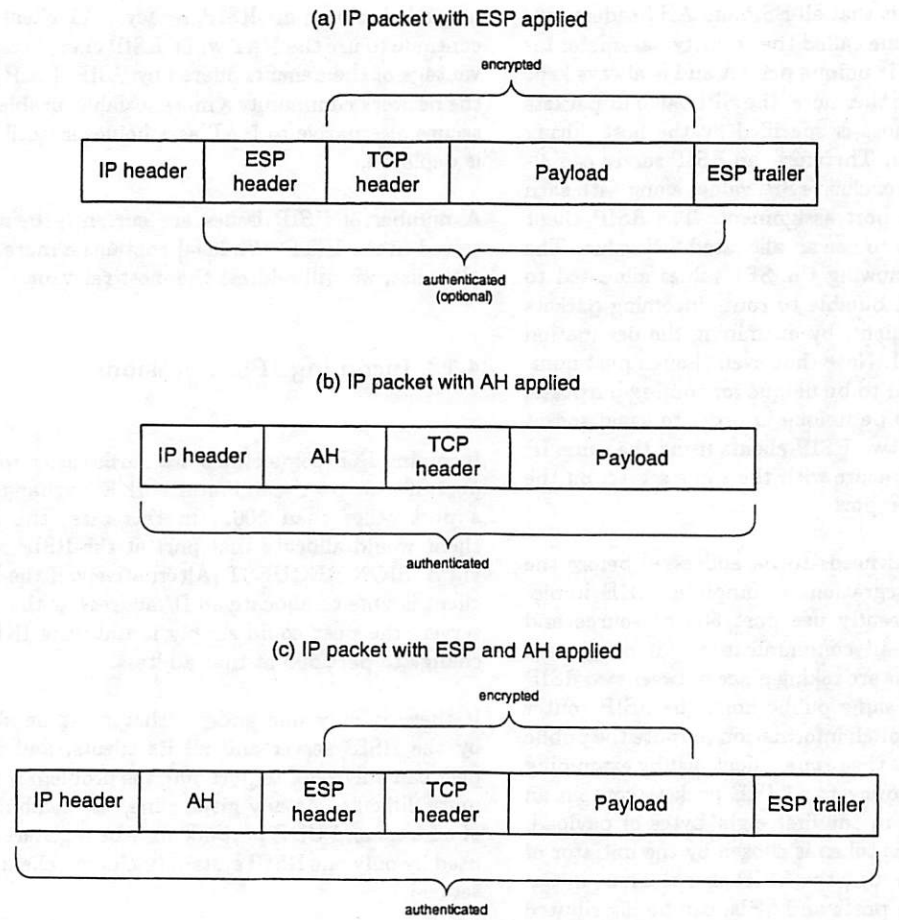

Figure 2: IPsec protection with ESP and AH.

mode includes the transport header. Commonly, the transport protocol is either TCP or UDP, in which case the transport headers include a pseudo header containing, among other things, the source and destination address fields in the preceding IP header. Thus, indirectly, ESP in transport mode renders the IP source and destination addresses immutable, and is broken by NAT gateways.

Given that the keying material used for the encryption and authentication can only be shared between the RSIP client and its public peer, the RSIP-enabled gateway will not be able to read TCP/UDP ports when ESP is used, and will not be able to modify the ports when ESP or AH is used. Thus, IPsec generally will not work through a NAT. However, ESP in tunnel mode is impervious to NATs, in that the outer IP header is not cryptographically protected. Hence, it is possible to modify the outside header without rendering the packet useless. In spite of this, NAT gateways have no surefire way of establishing the appropriate mapping to demultiplex these packets. Recent Linux implementations use temporal association to guess what the right mappings are, based on the assumption that outgoing packets are immediately followed by incoming traffic. And from this, it is possible to guess which client will be expecting the subsequent incoming IPsec packets.

In order to support IPsec through an RSIP-enabled gateway, we need to solve two problems: (1) finding one or more fields in the packet headers to use to demultiplex incoming packets to RSIP clients, and (2) a way of ensuring that these fields are unique per public IP address used for RSIP. The key to address-

ing these issues is that all ESP and AH headers contain a 32-bit value called the security parameter index (SPI), that is unique per SA and is always kept in the clear. Furthermore, the SPI value in packets received by a host is specified by the host during IKE negotiation. Therefore, an RSIP server can allocate mutually exclusive SPI values along with each IP address and port assignment. The RSIP client will tell its peer to use an allocated SPI value. The RSIP server, knowing the SPI values allocated to each client, will be able to route incoming packets to the proper client, by examining the destination address and SPI. Note that even though port numbers do not need to be unique for routing purposes, they do need to be unique in order to avoid socket collisions when two RSIP clients using the same IP address communicate with the same server, on the same destination port.

One more detail needs to be addressed before the RSIP/IPsec integration is complete. IKE implementations currently use port 500 as source and destination for all communication. If concurrent IKE negotiations are taking place between two RSIP clients and the same public host, the RSIP router will not have enough information to route the public host's packets to the proper client just by examining the headers. However, all IKE packets contain an initiator cookie in the first eight bytes of payload. The value of this token is chosen by the initiator of the IKE session; i.e., the RSIP client. Thus, initiator cookies, like ports and SPIs, can be distributed in a mutually exclusive fashion by the RSIP server. A more elegant alternative is to let IKE clients use an ephemeral source port number. An RSIP implementation would then be able to choose a locally-unique port that could be used to demultiplex incoming IKE replies.

The discussions of RSIP/IPsec interactions in this section are necessarily brief due to space constraints. A more complete presentation of these issues, including the RSIP signaling messages for IPsec, is found in [11].

4 Implications and Future Work

RSIP presents a potentially revolutionary concept that can be deployed in an evolutionary fashion. RSIP overcomes the difficulties of NAT in a way that can co-exist with NAT. As NAT gateways are upgraded to support RSIP, legacy NAT clients can continue to use the NAT while RSIP clients take advantage of the benefits offered by RSIP. RSIP gives the network community a more scalable, usable, and secure alternative to NAT as a holdover until IPv6 is deployed.

A number of RSIP issues are currently being resolved in the IETF. While [4] contains a more complete list, we will address the most relevant.

4.1 Incoming IPsec Sessions

Incoming IKE connections are much easier to support if the peer can initiate IKE exchanges to a port other than 500. In this case, the RSIP client would allocate that port at the RSIP server via ASSIGN_REQUEST. Alternatively, if the RSIP client is able to allocate an IP address at the RSIP server, the peer could simply initiate the IKE exchange to port 500 at that address.

If there is only one address that must be shared by the RSIP server and all its clients, and if the peer can only send to port 500, the problem is much more difficult. At any given time, the combination of address and UDP port 500 may be registered and used by only one RSIP system (including clients and server).

Solving this issue requires demultiplexing the incoming IKE connection request based on something other than the port and address combination. It may be possible to do so by first registering an identity with a new RSIP command of LISTEN_RSIP_IKE. Note that the identity could not be that of the IKE responder (the RSIP client), but that of the initiator (the peer). The reason is that IKE Phase 1 only allows the sender to include its own identity, not that of the intended recipient (both, by the way, are allowed in Phase 2). Furthermore, the identity must be in the clear in the first incoming packet for the RSIP server to be able to use it as a demultiplexor. This rules out all variants of Main Mode and Aggressive Mode with Public Key Encryption (and Revised Mode of Public Key Encryption), since these encrypt the ID payload.

The only Phase 1 variants which enable incoming IKE sessions are Aggressive Mode with signatures or with pre-shared keys. Because this scheme involves the RSIP server demultiplexing based on the

identity of the IKE initiator, it is conceivable that only one RSIP client at a time may register interest in fielding requests from any given peer. Furthermore, this precludes more than one RSIP client's being available to any unspecified peer.

Once the IKE session is in place, IPsec is set up as discussed in this document, namely, by the RSIP client and the RSIP server agreeing on an incoming SPI value, which is then communicated to the peer as part of Quick Mode.

The alternate address and port combination must be discovered by the remote peer using methods such as manual configuration, or the use of KX [12] or SRV [13] records. It may even be possible for the DNS query to trigger the above mechanisms to prepare for the incoming and impending IKE session initiation. Such a mechanism would allow more than one RSIP client to be available at any given time, and would also enable each of them to respond to IKE initiations from unspecified peers. Such a DNS query, however, is not guaranteed to occur. For example, the result of the query could be cached and reused after the RSIP server is no longer listening for a given IKE peer's identity.

Due to the limitations implied by having to rely on the identity of the IKE initiator, the only practical way of supporting incoming connections is for the peer to initiate the IKE session to a port other than 500.

4.2 General Disparate Address Space Support for RSIP

An RSIP server is located at the border between two disparate address spaces. In most deployments scenarios, this is the border between the global Internet and a private network. In other scenarios (for example in business to business communications), the address spaces at either side of the RSIP server may have conflicting address ranges. This may happen if both address spaces use net 10.0.0.0 within their network. In this case, the QUERY_REQUEST cannot be resolved by the RSIP server by just examining an IP address. A variant of QUERY_REQUEST that uses DNS names instead of IP addresses may solve this issue.

A general solution probably implies further refinements to the protocol.

4.3 RSIP as a IPv6 Transition Mechanism

In [1], the future of IPv6 deployment is addressed. Three scenarios are explored, one in which IPv6 never deploys, another in which it partially deploys, and a third in which it fully deploys. It is very unlikely that IPv6 will fully deploy without some intermediate form of partial deployment. Thus, a key ingredient for the transition to IPv6 is the existence of transition technologies [14] that allow IPv6 to be deployed on an incremental basis, while coexisting with the legacy IPv4 infrastructure.

In situations in which IPv6 is deployed on some edge networks while backbones and other edge networks remain IPv4-only, RSIP can play a valuable role. The hosts on the IPv6 edge networks may be dual stack (i.e., they simultaneously support both IPv4 and IPv6). By placing RSIP clients in these hosts and an RSIP server on the gateway router between the IPv4 and IPv6 spaces, RSIP can allocate IPv4 addresses to the IPv4 stacks of these hosts when necessary. For example, the following communications can be supported:

- *IPv4/IPv6 dual stack host communicating with another local IPv4/IPv6 dual stack host*: Use IPv6.
- *IPv4/IPv6 dual stack host communicating with another remote IPv4/IPv6 dual stack host*: Use IPv6 locally. When the local edge router receives the packets, it performs IPv6-over-IPv4 tunneling to the remote IPv6 edge network. The remote edge router terminates the tunnel and forwards the IPv6 packet to the destination host.
- *IPv4/IPv6 dual stack host communicating with a remote IPv4 host*: Use RSIP to acquire a public IPv4 address, and use that address to contact the remote IPv4 host. The IPv4 packets are transmitted on the IPv6 edge network using an IPv4-over-IPv6 tunnel, which is terminated at the local edge router.

This system is very similar in spirit to the Dual Stack Transition Mechanism [15] proposal, which uses DHCP for IPv4 address allocation. However, DSTM does not allow multiple clients to share the same IP address from the gateway machine. In

other words, DSTM does not have an equivalent to the RSAP-IP [9] or RSIPSEC [11] methods in RSIP.

5 Conclusion

History has shown us that changing the core of the network to support a new protocol is very difficult. The lack of widespread RSVP, IP multicast, and IPv6 deployment attests to this premise. However, changes to the network edge occur gradually over time. Privacy concerns have led to the deployment of firewalls and host security in almost every edge network. Configuration has been eased by deployment of DHCP. Roaming is supported by Mobile IP. RSIP is another way to upgrade the edge of the network to overcome the limitations of legacy network design.

References

[1] B. Carpenter, "Internet transparency." Internet RFC 2775, Feb. 2000.

[2] P. Srisuresh and M. Holdrege, "IP network address translator (NAT) terminology and considerations." Internet RFC 2663, Aug. 1999.

[3] S. Kent and R. Atkinson, "Security architecture for the Internet Protocol." Internet RFC 2401, Nov. 1998.

[4] M. S. Borella, J. Lo, D. Grabelsky, and G. Montenegro, "Realm Specific IP: Framework." Internet Draft draft-ietf-nat-rsip-framework-03.txt, Dec. 1999. Work in progress.

[5] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear, "Address allocation for private internets." Internet RFC 1918, Feb. 1996.

[6] J. Privat and M. S. Borella, "DHCP next server option." Internet Draft draft-ietf-dhc-nextserver-01.txt, Feb. 2000. Work in progress.

[7] E. Guttman, C. Perkins, J. Veizades, and M. Day, "Service location protocol, version 2." Internet RFC 2608, June 1999.

[8] J. Kempf and G. Montenegro, "Finding an RSIP server with SLP." Internet draft draft-ietf-nat-rsip-slp-00.txt, Feb. 2000. Work in progress.

[9] M. S. Borella, D. Grabelsky, J. Lo, and K. Taniguchi, "Realm Specific IP: Protocol." Internet Draft draft-ietf-nat-rsip-protocol-05.txt, Jan. 2000. Work in progress.

[10] D. Harkins and D. Carrel, "The Internet key exchange (IKE)." Internet RFC 2409, Nov. 1998.

[11] G. Montenegro and M. S. Borella, "RSIP support for end-to-end IPsec." Internet Draft draft-ietf-nat-rsip-ipec-02.txt, Feb. 2000. Work in progress.

[12] R. Atkinson, "Key exchange delegation record for the DNS." Internet RFC 2230, Nov. 1997.

[13] A. Gulbrandsen and P. Vixie, "A DNS RR for specifying the location of services (DNS SRV)." Internet RFC 2052, Oct. 1996.

[14] R. Gilligan and E. Nordmark, "Transition mechanisms for IPv6 hosts and routers." Internet RFC 1933, Apr. 1996.

[15] J. Bound, L. Toutain and H. Afifi, "Dual stack transition mechanism (DSTM)." Internet Draft draft-ietf-ngtrans-dstm-01.txt, March 2000. Work in progress.

Towards a Platform for Intelligent Activity at the Edge

Hilarie Orman
Novell, Inc.
horman@novell.com

Abstract

Proxy caches for content on the Internet are high-performance platforms with complex software services. Because they understand application semantics, and because they have a great deal of memory, they are the natural place for new services that are tailored closely to site or user preferences and requirements. The engineering aspects of caches and how they contribute to a new network infrastructure for highly capable or intelligent services are examined in this paper.

1. Introduction

From the beginning of the networking era, there have been expectations of intelligent software as integral part of applications. The concept of distributed software agents followed quickly on the heels computer networking, as evidenced by the early work on actors with distributed control [Hew77]. This work assumed widespread, inexpensive methods for communication and control, but no large-scale system emerged to meet the hopes of the intelligent agent researchers. Instead, the Internet thrived on applications with simple point-to-point semantic. As the Internet entered its more recent wildly successful phase, more thinking was invested in having the network software itself perform complex tasks. Two extreme viewpoints in this space have been intelligent agents and active networks. However, in both cases, the infrastructural support for the concepts has lagged far behind the thinking. What we have seen is the success of an Internet architecture that presents simple end-to-end semantics but with an implementation that has many more layers "in the middle". The thesis of this paper is that an emerging middleware layer, implemented on edge devices, provides the foundation for the addition of greater capability to large-scale networks, and this is part of the enablement of a new class of services.

2. Background and Overview of Proxy Architectures

As the Internet evolved from a small research experiment into a global communication infrastructure, the uniform architecture began to separate into two pieces: the high bandwidth core and the edge organizations (collections of LANs). Separate routing protocols evolved for the two regimes, and some differences in addressing and naming were smoothed over by translating gateways. Finally, security concerns led to a fairly complete separation enforced by firewalls.

The World-Wide Web brought about a revolution in the Internet by making it easy and inexpensive to publish and consume high-grade documents (visually pleasing, organized, etc.). Most Internet users came to believe that the Internet and the WWW were synonymous. Within a matter of a few years, the problem of scaling the web became a critical matter. To solve the problems of reliability and latency, engineers have been building a new infrastructure on the old frame.

The first important advances in the network were load balancing for WWW servers and caching to reduce latency in delivery. The platforms that assist in the first category are called "reverse proxies", and the platforms in the second category are "proxies." Both are critical in avoiding Internet meltdowns. For example, proxies are an essential component in quality of service, because the latency of Internet accesses depends on the square of the number "hops" [Rob98]. Proxy caches reduce the number of hops and thus contribute to improved latency; they also reduce the congestion in the core of the Internet and make other services run faster.

It was natural to add proxy caches to firewalls or security gateways, and this "platform sharing" has contributed to widespread adoption of both forward and reverse proxies. This has also provided a way to unify security services by collocating two different functions, both having common security dependencies, in one box.

The placement of such services at the point joining an organization is also an architectural "chokepoint" where engineering for high performance has a very good payoff. Competitions for high-speed cache services have been intense and have bred a new class of engineers who are expert in network stack scaling and optimization. Web caching workshops and competitive bakeoffs have become part of the Internet engineering landscape.

The second stage of development is going on today, as a middleware layer of cooperating content repositories and/or caches is deployed by content distribution services. In this phase, there is movement of content to distribution sites, again, to reduce latency due to network transmission times but also to reduce the latency due to the load on the origin servers.

Caches can be the platform for adding intelligent services to the network infrastructure. As we will see, they differ from switches and routers because they have much more RAM and disk space. Because so much storage is available on very fast, inexpensive machines, there are opportunities for adding much more in the way of application level services, invoked both explicitly and implicitly, and for keeping the state information that the services require.

The backing store for caching systems presents opportunities for optimization that are novel with respect to traditional network file systems such as NFS or AFS. The problem might be presented as "caching all the way down", because the disk system is primarily for holding the working set of the cache, and the consistency requirements for backing store are much relaxed from those of a file system. This flexibility offers more opportunities for minimizing disk access time by careful selection of cylinder groups and at the same time imposes fewer "housekeeping" requirements. One might consider the backing store to be similar to a file system that has frequent deletes with no processing cost for the deletion.

There are research areas open in the management of disk backing stores, keeping multiple copies of objects, coping with dynamic content, tighter synchronization with origin servers, and moving to a hybrid system that supports network file systems with efficient caching store.

3. Smart Middleware Boxes

3.1. Performance Management

Web caching applications are probably the most stressful environment for Internet protocols. Almost any performance anomaly will be magnified in such an environment, and engineers who understand the details of a successful networking implementation are being honed in greater numbers than ever before. To meet the demands of holding the working set of web data, the configuration of a caching server or proxy uses anywhere from 256M to 4G bytes of RAM and from a gigabyte to nearly a terabtye of disk storage.

The point of caching is, of course, primarily to reduce latency. The Internet suffers from great variations in latency because it does not reserve resources in advance of communication. The latency variation can be two or more orders of magnitude in common cases Kal[99], though particular point-to-point connections may show almost no variation for weeks at a time. The caches serve to minimize the latency and its variation, leading to a much more predictable service model.

Web caches that are placed near the end user (the browser) are frequently configured as "forward proxies", and they typically have very large working sets (millions of objects) in order to achieve a cache hit rate of over 50%. The latency variation can become much smaller when the cache is near the user, because the number of buffering points between the user and the data is probably only 2 or 3 (e.g., a router and a proxy cache).

Caches can also reduce latency by distributing the load from the original content site to the high performance cache. This offloading of network communication and object retrieval leaves the original content servers free to perform non-cacheable computations in support of their web service; the computations are usually hidden in scripts that are privy to confidential information and/or large databases. Caches that reduce latency by offloading the content have working sets which are are typically much smaller than the ones for forward proxies. In fact, at times all of the site's content fits into RAM on its reverse proxy; in contrast, a forward proxy may have trouble fitting the index for its disk-cached content into the same amount of memory.

Web caching systems have the latitude to implement efficient networking systems in ways that are difficult for general purpose operating systems. For a server machine to handle 10K connections and delivery of up to a 10^10 objects per hour is a difficult task for most network stacks. Such systems must have very fast TCP message dispatching, space for large amounts of connection state (the TCB's), and efficient handling of storage for network data. Work in the research community over the last decade has repeatedly demonstrated methods that help meet the goals; the use of polling in preference to interrupts [Min93], "no copy" I/O for network data, and the ability to communicate with applications without copying or re-encoding data [Dru93]. Because web caching systems do not have the burden of supporting general purpose processing as in standard operating systems, they can make use of the research results for very high throughput. Web caching systems today can handle a sustained rate of up to 24K requests/second, which far exceeds the rates considered maximal 10 years ago.

Name lookups, for the Domain Name System (DNS), can be a major bottleneck for high performance boxes, and most of these systems maintain huge caches of name-to-IP-address mappings, in effect caching much of the root server information and many common hostnames (because www.xxxzzzy.com is the usual form of a name presented for resolution). A cache of 20K domain names usually results in a 99% hit rate for web proxy systems; the number of instructions for a DNS lookup is little more than the time to hash the name and do the lookup. Cache misses are infrequent, occurring only once every few seconds, and this keeps UDP connections down to a very few. A 20K cache could be usefully configured to serve all DNS lookup needs for an entire organization.

3.2 Expanded functionality

The middleware layer of web caching boxes is rapidly expanding. As web caching appliances become more pervasive, more uses get loaded onto them. Thus, we see the next stage of the Internet architecture revolution: the movement of a service infrastructure into the network.

Some services already exist in the net. These include many kinds of web-based mail services, homepage hosting, file storage, messaging. However, for the most part, these are not distributed services, and they represent small points or islands of service, themselves separated by firewalls. Web caches, on the other hand, sit in the infrastructure itself, and their services apply to the applications riding above them.

New services that are emerging through this middleware are centered on web functions. We note several areas where current research and products are centered: object-specific functionality, user-specific functionality, and authentication and access control services.

3.2.1 Name Translation and Content Transcoding Intelligence

There has been no simple way to have content available from multiple sites in the Internet. Replication strategies, meant to reduce latency and to distribute server load, often founder on single points of provisioning (such as DNS servers). This is rapidly changing, and users now find that loading a web page from one location can mysteriously turn into loading a page from any of a number of servers seemingly unrelated to the target.

One way of doing this involves translating the content, turning an embedded name (URL) into a new name, perhaps by changing the hostname, based on the user's location, current conditions, or other factors. In essence, multiple path routing results from URL translation.

The ability to select a new server location and to translate the embedded names very rapidly is a level of intelligence not normally associated with applications. The devices that do this have streamlined ability to parse HTTP and HTML, to do text substitution, and to deliver the results to the network interface with minimal overhead. This is an edge service that requires some understanding of the application language, and it turns protocol layering concepts on their heads.

Novell's approach to web proxies uses translation for two applications. One is to assist in load balancing and another is to secure the transmission of confidential data. In each case, the translation is done by pattern matching on URL's.

A different mechanism retains state between object name references, establishing a user-specific mapping with a short lifetime. For example, an image and the site with information about the image might be linked implicitly. When the user browser requests the image, a computation at the web cache, acting as a proxy, determines the image and its related site. This makes the content of the page itself a function of the proxy, and this is an especially rich service model.

A more complex processing model, for services that are very computationally expensive or only available on non-proxy platforms, uses nearby processors that are in the same administrative domain as the proxy. These might be thought of as "lollipop" processors, because the processing flow, normally straight through the proxy between the client and server, now takes a tour sideways.

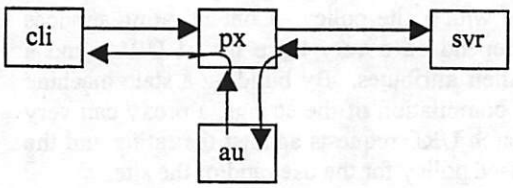

Figure 1: Example data flow, auxiliary processing using ICAP

The proposed protocol between the proxy ("px") and the auxiliary processor ("au") is built on HTTP and is known as the Internet Content Adaptation Protocol – ICAP [ICAP00].

Examples of complex services that might be performed by an auxiliary processor include: having HTML text converted to different human languages, audio content transcoded to achieve a service level based on current resource availability (network quality of service), negotiation with advertisers for micropayments for accepting advertising and offering demographics, access to service information that is proprietary to the owner of the auxiliary processor box, personalization of web pages, hosting user processes for directed personalization, management of "persona" information based on user policy and profiles, analysis of access patterns to remote or local web servers (correlation, clustering, neural network creation), access to trusted financial services, high assurance security services, etc.

3.2.2 Object Policy

If, by abuse of language, we consider URL's to be object names, we can say that edge devices are delivering a form of object intelligence by analyzing object names and mapping them to attributes. These attributes may be simple, such as content ratings based on external evaluations, or quite complicated, such as automatically generated content classification based on word or image analysis, cluster analysis based on access patterns by other users, etc.

Architecturally, the ability to map names to attributes and evaluators is an important advance in the capability of edge devices. It entails maintenance of state, caching, and parsing of names at a chokepoint between the end consumers and their network access. The mechanics that allow this are a general solution for several classes of problems, and can be the foundation for performing such functions as selecting the image resolution to be used for low-bandwidth or low-power devices in the future.

Access control based on content rating is a discretionary policy generally set at a forward proxy by users or on their behalf by the proxy administrator in accordance with a site policy. Content rating services typically depend on a very large list of URL's and a vector of their attributes. By building a state machine that is the compilation of the strings, a proxy can very quickly match URL requests against the rating and the content-based policy for the user and/or the site.

The performance cost of the policy check is quite small; the number of instructions executed is approximately equal to the number of bytes in the URL name. The main performance penalty comes from the amount of space used by the state machine; that memory is not available for caching of content, and this leads to the biggest performance penalty in terms of content delivery. When the policy database occupies 10% of the memory, observed performance on web benchmarks decreases by about 5%, even if the database lookups are not executed. If the lookups are executed, they only use about 5% of the total machine cycles.

The efficiency of a cache depends on the available memory. For a forward proxy, the ratio of disk to RAM can be as high as 10K to 1. The index for the disk cache can use nearly as much memory as the objects cached in RAM, in extreme cases. The RAM requirements for classification of hundreds of thousands of URLs is a noticeable fraction of the memory, and this mandates that any auxiliary data associated with URL sets be represented as compactly as possible. It is possible to maintain auxiliary data for hundreds of thousands URLs that are non-resident in the cache and to access the data quickly when making a decision about whether or not to fetch a page, what quality of service to use for its transfer, or what origin server to use for its retrieval.

The dependence of performance on memory demonstrates a point of importance for intelligent edge devices: in general, proxy caches are not CPU limited, running idle as much as 90% of the time in normal operation. This leaves a lot of cycles for running computational tasks on the caching system itself.

3.2.3 Identity and Security

Edge devices are the injection point for basing access control decisions on Internet-wide identities and for management of identity information. Most large organizations have fine-grained management of access control internally, but only the coarsest management of objects presented externally. A common policy for many organizations is that the only bi-directional information flow is through SMTP, and the only HTTP access is for information that is cleared for public access. In the latter case, the information goes onto a web server that is primarily "read only".

The SSL3 protocol [Fre96] is the *de facto* standard for secure connections for web browsers today, but while it is sufficient for encrypting TCP connections, there are few systems that integrate it into authorization or access control. By clever use of the HTTP protocol, it is possible for edge devices to interpose access control mechanisms based on SSL privacy and authentication.

Reverse proxies, sitting near the authoritative sources for content, are natural enforcement point for access control at the granularity of a web page. They can

present an access policy for the outside world that is a consistent extension of the internal controls, without changing any of the legacy software for the origin site. The reverse proxy can implement identity services, authentication services, and access control enforcement on behalf of the origin server organization, using SSL as the common security protocol.

Consider an organization with a public web server that wants to enforce a simple security model: information on its server www.myorg.com is public (no authentication), while information on www.internal.myorg.com is accessible to any user authenticated by the network authority for myorg.com. Novell's reverse proxy solution to implementing this policy allows the administrator of the reverse proxy to enter the control rules as initial configuration schema for the proxy, to specify a certificate for authenticating the server to the clients, and to designate an authentication server for trusted communication.

When the reverse proxy receives a request for an item in www.myorg.com, it fetches the object from the origin server if need be (communication between the proxy and the origin server is trusted) and delivers the page to the requestor. If the client request is for an object in www.internal.myorg.com, perhaps '/plan1.html', then it redirects the client to use https as the access protocol. The client browser will comply with an SSL encrypted connection to the reverse proxy, and the proxy will redirect the client to an authentication server, and that server may either use SSL mutual authentication to establish an authenticated identity or it may redirect the client through a longer login dialogue. At the end of the dialogue, the reverse proxy will receive an indication from the authentication server that the login was either successful or unsuccessful. In the former case, it will deliver the originally requested object to the client over their SSL connection; the latter case it will respond to the client with an error message.

The use of SSL requires no changes to either the client or server and illustrates how a new service is easily built on a proxy cache foundation. Obviously, more complicated and finer-granularity access control policies are easily added as the requirements of the organization evolve.

If authentication intelligent edge devices are the directors of the authentication process, they must have the intelligence to deal with identity information. This is an attractive notion, because a trustworthy platform in the network avoids the many problems associated with maintaining security information on end-user machines or bound to particular organizations. Internet users are beginning to desire an Internet persona that is under their own control, even when their employment or Internet service provider changes.

An even more open model would allow the objects to have access control lists with pairs consisting of the distinguished name and certifying authority of authorized users. This allows organizations to extend their fine-grained control methods to the growing class of Internet users with credentials backed by public keys. While it may be difficult to retrofit legacy operating systems with such controls, proxies that have been extended to understand identity, authentication, and access controls can easily enforce access rules that are extended far beyond the legacy model.

4. Next stage: the Intelligent Infrastructure

Adding more intelligent services to the network edges is a key part of creating new network services and of scaling the performance requirements to encompass global services. We will continue to see more software added to edge devices in the future. Routers, switches, proxy services, and other edge dwellers will creep "outwards" towards application services. The way to keep the network healthy during this transition is to work towards an architectural model that supports programming at the network edges.

Edge devices today are specialized creatures stepping gingerly towards open standards, common application interfaces, and extensibility. The edge world can take a giant step forward by defining some common execution environments and library services for unifying the services. A programmability model could take a two or three step approach by implementing a series of execution environments that have engineered tradeoffs of expressibility vs. ease of use vs. security.

The first tier programming language would define a language based on the keywords or tags and a set of transformation rules based on regular expression replacement operations. API's for common protocols such HTTP, HTML, SNMP, RTP, etc. would be available as an open standard. These rules would be dynamically loadable and compilable on receipt.

The second tier of languages would be bounded time execution languages with variables and assignments and conditionals, each with a security policy enforced by a policy engine in the execution environment. These languages would also be dynamically loadable and amenable to just-in-time (JIT) compilation.

The third tier languages would be general purpose and dynamically loadable; Java is a good candidate for a third-tier language. Third tier languages present security challenges, but their expressive power is a compelling force, and operating systems with effective process isolation mechanisms will be able to run them at the network edges with minimal risk.

With a programmable network edge, application semantics that operate on network data "flows" are possible, and they can assure consistent semantics for applications even when the applications are resident in the edge infrastructure, rather than at a central site.

The next challenge is to move from explicit programming of the network edge to programming that flows transparently to the points in the network that can most effectively execute the application.

The original concept of the Internet's end-to-end semantics for data transport and complex semantics for applications, is yielding to a richer model that allows edge devices to play a part in routing, caching, and executing application semantics. This is an evolving model, moving through a logical series of steps towards more complicated distributed execution models.

The client-server model, based on RPC [Whi76], has been a very successful way to build network services. The stateless semantics simplify the design of the client, and HTTP in large part relies on an RPC model. HTTP proxies are able to execute the semantics "parasitically", i.e., in the case of transparent proxies, they are not the intended ("addressed") parties to a transaction, but they can fulfill it nonetheless. Proxies have difficulty participating in connections where the server generates responses by executing local code with semantics that are unavailable to the proxy (e.g. cgi-bin scripts) or where client runs code from the server (e.g. "applets"). Active networks [Ten98], use an execution model that bases transport services on code that is executed through the infrastructure. Proxy platforms can enable a happy medium in which applications are either written with explicit proxy code that moves some of the application functionality onto proxy platforms ("proxylets" [Cro00]), or else the applications use servlets or applets with well-understood semantics that can be parasitically executed by the proxies. Both forward and reverse proxies can participate, based perhaps on the available of environmental information.

The logical extension of the distributed execution model may support mobile code, mobile agents, or other "untethered" applications that are not feasible today. The platforms for running these may resemble proxy platforms but they will have access to a broader range of standard semantics, directories, verifiable credentials, and most likely a computing base that supports higher assurances of trust than today's machines.

A third stage of evolution that an intelligent infrastructure enables is third-party services that exist for the purpose of linking other services together on behalf of users. Today consumers can invoke shopping agents that attempt to maximize the value of information to the user, but the agents must execute in adversarial environments where their goals are thwarted. An intelligent service infrastructure will enable user agents to negotiate in good faith with brokering services in order to get a broad spectrum of information with higher value. These services will meet and execute "in the network".

Engineering the next generation of network services will necessitate an intelligent infrastructure. Challenges exist in security, extensible service models, and interoperation in new areas such as the integration of telephony and Internet services with wireless handheld devices, the next generation of Internet video services, collaborative document management, and the instant messaging services hovering in our future. In order to make these services scalable at the same pace that we have seen the Internet and WWW services flourish, a programmable distributed service infrastructure is an obvious architectural requirement.

The infrastructure itself may likely yield a revolution much like we have seen in web services. This is likely to be a software agent infrastructure, with "safe enclaves" existing in the communications infrastructure itself. With this, users will be able to launch their software shopping agents into the network for untethered operation and trustworthy results.

5. Conclusions

Web caching and proxying systems are developing the platforms for an important new class of application accelerators and enhancers at the network perimeter. While the core of the network can be dedicated to moving bits over increasingly "fatter" pipes for more and more multimedia data, the need to dynamically manage information for individuals and businesses is driving the edge devices to a rich services model. This is ushering in a new notion of what constitutes the essential services of a global internet, of who engineers it and how, and it blurs the line between application and communications infrastructure. This trend is likely to continue, and it illustrates the mechanisms that will

continue transforming the Internet through the next few decades.

Bibliography

[Cro00] Crowcroft, John, Self Organising Application-level Routing, Tech Report RN/00/23, University College London, Feb. 2000.

[Dru93] Peter Druschel and Larry L. Peterson, Fbufs: A High-Bandwidth Cross-Domain Transfer Facility (1993), ftp://ftp.cs.arizona.edu/reports/1993/TR93-05.ps

[Hew77] Hewitt, C. (1977), "Viewing Control Structures as Patterns of Passing Messages", Artificial Intelligence 8(3), 323-364.

[ICAP00], Elson, J. et al, ICAP the Internet Content Adaptation Protocol, http://www.i-cap.org/specification.txt

[Kal99] Kalidindi, Sunil, and Zekauskas, Matthew J., Surveyor: An Infrastructure for Internet Performance Measurements, INET '99 Proceedings

[Fre96] Freier, Alan O. and Karlton, Philip and Kocher, Paul C. The SSL Protocol Version 3.0, Mar. 96, http://www.netscape.com/eng/ssl3/ssl-toc.html,

[Min94] Minshall, Greg, and Major, Drew and Powell, Kyle, An Overview of the NetWare Operating System, Proceeding of Usenix Winter Technical Conference, 1994.

[Rob98] Roberts, Larry, Plenary Address, SIGCOMM '98, Vancouver, CA.

[Ten97] Tennenhouse, D, et al., A Survey of Active Network Research, IEEE Communications, Jan. 1997

[Whi76] White, J.E., A high-level framework for network-based resource sharing, Proceedings of the National Computer Conference, June 1976.